£1.00

ELBUG FOR THE ELECTRON

Vol 1 No 10 October 1984

Digger

Games

- **Digger**
- **The Memory Game**

PLUS

- **Faster Programming**
- **Compacting Basic Programs**
- **Fireworks Display**
- **Cassette Troubles**

PLUS

- **Latest Add-ons Reviewed**
- **New Games Reviewed**
- **Book Reviews**
- **And much more**

EDITORIAL

THE FIRST YEAR OF ELBUG

With this issue of ELBUG we complete the first volume of 10 issues of our magazine for Electron users. Each month we have striven to provide a variety of material to entertain you and to assist you in making the most of your Electron. The Electron is now in good supply in the shops, and there are a growing number of different add-on units and a wide range of games and other software now available. We believe that Acorn will soon announce details of their Plus-3 module to complement the Plus-1 released earlier this year. This will allow disc units to be used with the Electron, giving much improved saving and loading of files compared with cassette.

We are preparing a useful index to all the issues of volume 1 and this will be included free with your next copy of ELBUG (Vol.2 No.1). You will have seen from the previous issue that we now have available an attractive binder to contain all the issues of volume 1, and there is provision to include the index as well. The next issue of ELBUG will be the Christmas issue, so you can expect an entertaining and seasonal magazine.

THIS MONTH'S MAGAZINE

This issue sees the publication of the last part in our series on Electron Graphics. We have tried to finish with something of a flourish by including no less than five interesting examples of graphics programs. We hope you have found this series both useful and entertaining, and no doubt we shall be returning to the fascinating world of graphics in future issues of ELBUG.

Amongst the letters that we receive from ELBUG members, problems with the use of cassettes feature quite prominently. If you have troubles when using cassettes then you should find the article on this subject by Peter Rochford to be of great practical help.

We have also included in this issue another most useful utility in the form of a program compacter, which will shrink a larger program so that it will fit into the available memory. This is particularly useful when writing programs that use one of the 20K graphics modes (modes 0 to 2), and can sometimes make the difference between a working and non-working program.

There are lots more articles and programs in this issue of ELBUG plus reviews of all the latest software, hardware add-ons and books all helping you to make the most of your Electron.

Mike Williams

NOTICE BOARD NOTICE BOARD NOTICE BOARD NOTICE BO

Magazine Cassette

All the programs in this month's magazine are available on cassette and this month we have added two extra programs, the winning entry in the by Dave Channing in the'Oddfactors' Brainteaser competition (set in the June supplement), and a machine code action game, Astro Wars, by Alan Malik. This makes the magazine cassette even better value this month, and you can get one cassette free if you take out a subscription now (see the back cover for details).

Hint Winners

This month the £10 prize goes to P.Jollyman and the £5 prize to E.Westhead. Any new hints will always be most welcome.

ELBUG MAGAZINE

GENERAL CONTENTS

Page Contents

HINTS, TIPS & INFO

PROGRAMS

FIREWORKS DISPLAY

by D.D. Harriman

The short program listed here, by D.D. Harriman, is a good example of an animated graphics display. As such it is an excellent demonstration of the use of the VDU19 instruction described in this month's "Electron Graphics" article, and provides a colourful animation of a cluster of exploding fireworks.

To use the program, just type it into your Electron, and run it. The display uses mode 2 to build up a pattern of coloured dots. After a short delay the program will then animate the display to produce a multi-coloured sequence of exploding fireworks.

The fireworks program listed here is both brief and quite simple. A short section of code (contained in the procedure PROCF) displays a series of dots on the screen in different colours to represent three different types of firework (lines 160 to 180, 190 to 210 and 220 to 260). The colours are then redefined in a fixed sequence (using the procedure PROCPP called repeatedly at line 270) to produce the impression of movement.

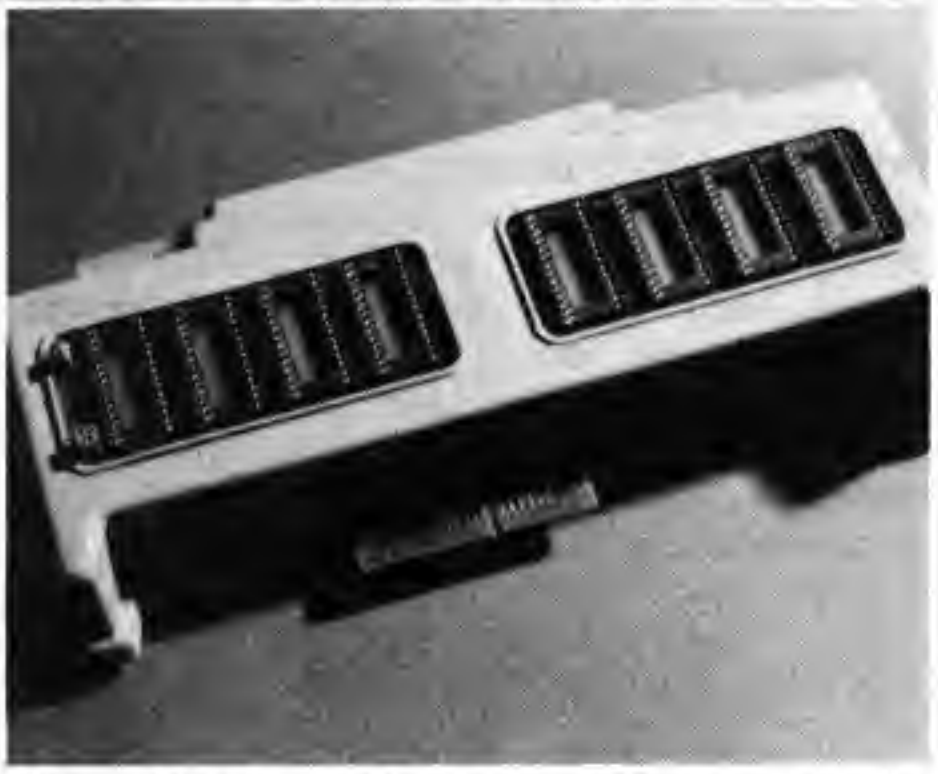

```
 10 REM PROGRAM FIREWORKS
 20 REM AUTHOR D.D. HARRIMAN
 30 REM VERSION E1.0
 40 REM ELBUG OCTOBER 1984
 50 REM PROGRAM SUBJECT TO COPYRIGHT
 60 :
100 ON ERROR GOTO 300
110 MODE 2:COLOUR143:COLOUR14
120 VDU19,15,0;0;:VDU19,14,7;0;12
130 VDU23,1,0;0;0;0;
140 Q%=0:PROCF(300,0,0,100,9,
13,0)
150 X2%=X1%:Y2%=Y1%:F2%=F1%
160 FOR A=0 TO PI*2 STEP PI/8.5
170 PROCF(X2%,Y2%,COS A*50,SI
N A*50,9,999,F2%)
180 NEXT:Q%=1
190 FOR A%=-8 TO 8
200 PROCF(900,0,A%,70+RND(10)
,5,999,0)
210 NEXT:F2%=0
220 FOR A=0 TO PI*2 STEP PI/7
230 GCOL0,F2%:MOVE600,800
240 DRAW600+COS A*50,800+SIN
A*50
250 PROCF(600,800,COS A*50,SI
N A*50,9,999,F2%)
260 F2%=F2%+1:NEXT
270 REPEAT PROCPP:UNTIL INKEY
0<>TRUE
280 END
290 :
300 MODE 6
310 ON ERROR OFF
320 IF ERR<>17 REPORT:PRINT"
at line ";ERL
330 END
340 :
1000 DEF PROCW
1010 TIME=0:REPEAT UNTIL TIME>3
1020 ENDPROC
1030 :
1040 DEF PROCPP
1050 C%=RND(7):FOR F%=0 TO 13
1060 PROCW:VDU19,F%,C%;0;19,(F
%+13)MOD14;0;0
1070 NEXT
1080 ENDPROC
1090 :
1100 DEF PROCF(X%,Y%,XM%,YM,G,
N%,F%)
1110 FOR K%=1 TO N%
1120 X%=X%+XM%/1.5:Y%=Y%+YM/1.5
1130 YM=YM-G:F%=(F%+1)MOD14:GC
OL0,F%
1140 PLOT69,X%,Y%
1150 IF X%>-1 AND X%<1280 AND
Y%>-1 AND Y%<1024 NEXT ELSE K%=
9999:NEXT
1160 X1%=X%:Y1%=Y%:F1%=F%
1170 ENDPROC
```

NEW ELECTRON ADD-ONS

Reviewed by Alan Webster

This month sees the emergence of three new add-ons for the Electron, a ROM expansion board, a printer interface and a combined printer and joystick interface.

Product : Slogger ROMBOX
Supplier : Slogger Advanced Systems, 215 Beacon Rd., Chatham, Kent. ME5 7BU.
Price : £39.50 incl. VAT



The first of the add-ons to be reviewed this month is a ROM expansion box from Slogger Advanced Systems which will hold up to eight extra ROMs. ROM based software has already proved very successful for the BBC micro and the story is likely to be the same for the Electron now that these add-ons are available. The box is made of moulded plastic in a darker shade of cream than the Electron. It is a pity that a little more trouble was not taken in the design of the ROMBOX to ensure a better match between the two units.

The unit plugs onto the edge connector at the back of the Electron and has two plastic screws to ensure a firm fixing. Although the ROMBOX has its own edge connector for further expansion, this does not provide any screw holes for the firm attachment of any such add-ons. The unit is supplied with an eleven page A4 manual, and contains very detailed instructions on the connection of the ROMBOX and the fitting of ROMs and RAMs.

This type of expansion is bound to become more popular as software companies start to produce ROM-based software for the Electron. Some software already exists on ROM such as Slogger's STARMON which is a machine code monitor. Other ROMs for the BBC micro will work partially on the Electron provided that they are not dependent upon the use of function keys and mode 7. Examples are Acornsoft's BCPL, Printmaster and Graphics ROM from Computer Concepts, and BEEBUGSOFT's Exmon and Toolkit, both of which will be available in Electron versions soon (watch the supplement for further announcements).

The ROMBOX is a sound product that will provide the necessary expansion to accomodate ROM based software as it becomes available. The ROMBOX also provides a connector for further expansion. Many potential purchasers are, however, likely to be put off by the rather crude appearance compared with the Electron itself.

Product : Printer Interface
Supplier : First Byte, 10 Castlefields, Main Centre, Derby. DE1 2PE.
Price : £34.95 incl. VAT

First Byte were one of the earliest companies to produce an add-on for the Electron when they produced their joystick interface (reviewed Vol.1

No.7). This printer interface is identical in appearance, with everything enclosed in a very neat moulded plastic case. As with the joystick port, there is no means of any further expansion.

The printer interface is packaged in a glossy and colourful box, and the instructions for the use of the interface are also printed on the packaging. All printers using a standard parallel Centronics interface should work with the Electron using this interface. The printer is readily controlled from the facilities already built into the Electron's operating system.

As with the joystick interface, First-Byte are to be congratulated on producing such a good looking and well designed product.

Product : PRINT STICK
Supplier : SIR Computers Ltd.,
91 Whitchurch Rd.,
Cardiff. CF4 3JP.
Price : £45.95 incl. VAT

The PRINT STICK from SIR Computers Ltd follows their ROM board which was reviewed in ELBUG Vol.1 No.5.

In appearance, PRINT STICK is quite different from the previously released ROM board, consisting of a fairly large flat pack that is a push fit onto the Electron's rear edge connector. Moulded in black plastic the interface provides a parallel printer connector and two switch-type joystick ports (Atari style joysticks) similar to the First-Byte joystick interface.

The unit contains its own software to provide the user with two extra commands (*DEFINEKEYS and *SCREENDUMP) providing useful and convenient control of the interfaces provided. The keys equivalent to joystick directions are defined using the first command. This allows almost any suitable game or other software to be quickly adapted to joystick control. For example if your game used the keys Z, X, *, ?, and RETURN for left, right, up, down and fire, then you would define the joystick to be equivalent to these keys.

The interface also contains a screen dump for a printer, including shading if the picture is in colour. This is activated by the command *SCREENDUMP, or by pressing Func and Copy together, and is a most useful feature of this device.

Overall, this interface is very well made and obviously a lot of thought has gone into its functional design. The built-in software functions are an excellent feature. All this is marred by the rather 'cheap and nasty' appearance of the black plastic casing which again does not match well in appearance with the Electron.

HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS

ELECTRON CLOCK IMPRESSION - R. Angus

The following program will enable an Electron to do an impression of a clock. It uses the cassette motor relay to produce the ticking noise. However, we don't advise you to leave your Electron ticking away or you might wear the relay out!

```
10  REPEAT TIME=0:*M.1
20  REPEAT UNTIL TIME>100:TIME=0:*M.0
30  REPEAT UNTIL TIME>100:UNTIL 0
```

DIGGER

by Andrew Logan

Digger is an arcade style game based on the popular home micro game of 'Monsters' or 'Panic!'. It is a one player game in which you have to kill the gremlins before they kill you, and I am sure you will find this an amusing and challenging game.

On running the program, you are presented with instructions on how to play and the keys to use. After pressing the space bar, the screen is drawn and the game starts. The screen consists of a number of walkways made from brick with several different height ladders connecting each floor.

The game starts with you being chased by three gremlins. To kill the gremlins you must dig a hole in the floor by pressing the space bar three times. When a gremlin falls into the hole you must hit it on the head with your shovel (by pressing the space bar again) so that it falls through to the next floor and dies. As you progress to the next level, you must dig two holes directly underneath each other and drop the gremlin through both in order to kill it. On the third level the gremlins have to be dropped through three levels and so on.

To make the game harder, you only have a limited amount of oxygen which slowly runs out while you play each level. If your oxygen runs out then you suffocate and the game ends. You start the game with three lives, and lose a life every time you are caught by a gremlin.

The keys to use for playing the game are 'Z' and 'X' for left and right and '*' and '?' for up and down. The spacebar is used to dig and to hit the gremlins on the head.

```
 10 REM PROGRAM DIGGER
 20 REM VERSION E0.4
 30 REM AUTHOR  ANDY LOGAN
 40 REM ELBUG   OCTOBER 1984
 50 REM PROGRAM SUBJECT TO COPYRIGHT
 60 :
100 ON ERROR GOTO 3180
110 MODE1:PROCtitle
120 MODE5:PROCstart
130 S%=0:U%=1:H%=3
140 PROCinit
150 PROCplatform
160 PROCladder
170 PROCsu
180 TIME=0
190 REPEAT
200 PROCm:PROCg
210 PROCscore
```

→

```
  220 UNTIL DEAD% OR CO%=3
  230 IF BON%<=0 GOTO260
  240 IFDEAD% ANDH%<>0 CLEAR:CLS:GOTO140
  250 IFCO%=3 U%=U%+1:CLEAR:CLS:GOTO140
  260 COLOUR3:PRINTTAB(6,18)"GAME OVER"
:FORT=0TO3000:NEXT:*FX15
  270 G=GET:CLEAR:CLS:GOTO 130
  280 END
  290 :
 1000 DEFPROCtitle
 1010 COLOUR3:PRINTTAB(12,2);"D I G G E
 R":COLOUR1:PRINTTAB(12,3);STRING$(11,"
*")
 1020 COLOUR2
 1030 PRINT'TAB(2);"Lure the GREMLINS i
nto the holes that"''"you dig in the br
ick-work and knock"''"them on the head.
To kill a Gremlin you"''"must knock it
through at least the"''"sheet number of
 levels.Kill all three"
 1040 PRINT'"Gremlins before the oxygen
 runs out if"''"you are to continue."''
''"Keys to use are:"''"Z - Left, X - Rig
ht, * - Up"''"? - Down, and SPACE to di
g or knock."
 1050 COLOUR3:PRINTTAB(12,30);"PRESS SP
ACE":REPEAT:UNTIL INKEY-99:ENDPROC
 1060 :
 1070 DEF PROCinit
 1080 IFU%=9 U%=1
 1090 CO%=0:OXY%=50+(U%*30)
 1100 IFU%>=4 OXY%=OXY%+150
 1110 LA%=FALSE:STILL%=FALSE
 1120 DIMA%(20,31),E%(5),F%(5),M$(3),PD
$(2),PRL$(2),B$(2),TRAP%(5),T%(5),DM%(5
),SG%(4)
 1130 DEAD%=FALSE:R%=0
 1140 FORI%=1 TO 3:M$(I%)=CHR$(I%+232):
NEXTI%
 1150 L$=CHR$231:P$=CHR$230
 1160 MD$=CHR$239:PRL$(1)=CHR$232:PRL$(
2)=CHR$237
 1170 PD$(2)=CHR$238:PD$(1)=CHR$242
 1180 G$=CHR$236:B$(1)=CHR$240:B$(2)=CH
R$241
 1190 SPL$=CHR$243
 1200 ENDPROC
 1210 :
 1220 DEFPROCstart
 1230 VDU19,1,6,0,0,0
 1240 VDU23,1,0;0;0;0;
 1250 VDU23,230,119,119,0,238,238,0,119
,119
 1260 VDU23,231,129,129,129,255,129,129
,129,255
 1270 VDU23,232,0,0,0,4,7,4,0,0
 1280 VDU23,233,24,24,0,124,190,25,36,34
 1290 VDU23,234,24,24,0,62,125,152,36,68
 1300 VDU23,235,90,90,66,126,126,36,36,
36
 1310 VDU23,236,231,36,60,126,219,126,3
6,60
 1320 VDU23,237,0,0,0,32,224,32,0,0
 1330 VDU23,238,0,0,0,0,128,80,32,64
 1340 VDU23,239,0,0,0,0,8,139,139,255
 1350 VDU23,240,0,0,0,238,238,0,119,119
 1360 VDU23,241,0,0,0,0,0,0,119,119
 1370 VDU23,242,0,0,0,0,1,10,4,2
 1380 VDU23,243,129,66,36,0,0,36,66,129
 1390 A%=0:REM SET HI-SCORE
 1400 ENVELOPE1,1,68,10,-127,240,113,14
,126,0,0,-126,126,126
 1410 ENVELOPE2,0,80,-110,-50,159,250,1
90,126,0,0,-126,126,126
 1420 ENVELOPE3,1,0,0,0,0,0,0,126,0,0,-
126,126,126
 1430 ENDPROC
 1440 :
 1450 DEF PROCplatform
 1460 COLOUR2:CLS
 1470 FORJ%=5TO29 STEP3
 1480 FOR I%=0TO 19
 1490 PRINTTAB(I%,J%);P$;:A%(I%,J%)=-1
 1500 NEXTI%
 1510 NEXTJ%
 1520 ENDPROC
 1530 :
 1540 DEF PROCladder
 1550 C%=0
 1560 COLOUR1
 1570 FORJ%=4 TO 25 STEP3
 1580 C%=C%+1
 1590 IFC%=3 C%=1
 1600 FOR L%=J% TO J%+3
 1610 IFC%=1 PRINTTAB(2,L%);L$;:A%(2,L%
)=2:PRINTTAB(10,L%);L$;:A%(10,L%)=2:PRI
NTTAB(17,L%);L$;:A%(17,L%)=2
 1620 IFC%=2 PRINTTAB(6,L%);L$;:A%(6,L%
)=2:PRINTTAB(14,L%);L$;:A%(14,L%)=2
 1630 NEXTL%
 1640 NEXTJ%
 1650 FORI%=1TO8
 1660 LX%=(RND(4)*4)-2:LY%=(RND(6)*3)+5
 1670 FORJ%=LY% TOLY%+2
 1680 PRINTTAB(LX%,J%);L$;:A%(LX%,J%)=2
 1690 NEXTJ%:NEXTI%
 1700 ENDPROC
 1710 :
 1720 DEF PROCm
 1730 IF DEAD% ENDPROC
 1740 N%=X%:M%=Y%:W%=B%:Q%=C%
 1750 IFINKEY-73 Z%=3:Y%=Y%-1:GOTO1810
 1760 IFINKEY-105 Z%=3:Y%=Y%+1:GOTO1810
 1770 IFINKEY-98 Z%=1:X%=X%-1:GOTO1810
 1780 IFINKEY-67 Z%=2:X%=X%+1:GOTO1810
 1790 IFINKEY-99 PROCdig:ENDPROC
 1800 STILL%=TRUE:ENDPROC
 1810 STILL%=FALSE
 1820 IFZ%=3 ANDA%(X%,Y%)<>2 X%=N%:Y%=M
%:ENDPROC
```


```
1830 IFA%(X%,Y%+1)=0 PROCfall:ENDPROC
1840 IFZ%=1 ANDX%<1 X%=1
1850 IFZ%=2 ANDX%>18 X%=18
 1860 IFZ%<>3 ANDX%<>1 ANDX%<>18 SOUND1
,2,185,1
 1870 IFZ%=1 B%=X%-1 ELSEIFZ%=2 B%=X%+1
 1880 C%=Y%
 1890 PROCprint
 1900 IFA%(X%,Y%)=4 ORA%(X%,Y%)=5 ORA%(
X%,Y%+1)=6 ORA%(B%,C%)=4 ORA%(B%,C%)=5
PROCdead
 1910 ENDPROC
 1920 :
 1930 DEFPROCprint
 1940 PROCback(N%,M%)
 1950 PROCback(W%,Q%)
 1960 COLOUR3:PRINTTAB(X%,Y%);M$(Z%)
 1970 IFZ%=3 ORLA% ENDPROC
 1980 COLOUR2:PRINTTAB(B%,C%);PRL$(Z%)
 1990 ENDPROC
 2000 :
 2010 DEFPROCg
 2020 IFDEAD%ENDPROC
 2030 R%=R%+1:IFR%>3 R%=1
 2040 K%=E%(R%):L%=F%(R%)
 2050 IFDM%(R%)=TRUE:ENDPROC
 2060 IFNOTSTILL% SG%(R%)=FALSE
 2070 IFTRAP%(R%)ANDTIME-T%(R%)>300 TRA
P%(R%)=FALSE:A%(E%(R%),F%(R%))=-1:COLOU
R2:PRINTTAB(E%(R%),F%(R%));P$:F%(R%)=F%
(R%)-1:E%(R%)=E%(R%)+SGN(X%-K%):GOTO215
0 ELSEIFTRAP%(R%):ENDPROC
 2080 IFSG%(R%)GOTO2100
 2090 IFSGN(X%-K%)=0 ANDINT((L%-1)/3)=(
L%-1)/3 SG%(R%)=TRUE:IFRND(1)>.5 P%=1 E
LSEP%=-1
 2100 IFSG%(R%)ANDA%(K%,L%+SGN(Y%-L%))<
>2 PROCstill:GOTO2150 ELSEIFSG%(R%)SG%(
R%)=FALSE
 2110 IFL%>Y% ANDA%(K%,L%-1)=2 F%(R%)=F
%(R%)-1:GOTO2150
 2120 IFL%<Y% ANDA%(K%,L%+1)=2 F%(R%)=F
%(R%)+1:GOTO2150
 2130 IFINT((L%-1)/3)=(L%-1)/3 E%(R%)=E
%(R%)+SGN(X%-K%):GOTO2150
 2140 ENDPROC
 2150 IFA%(E%(R%),F%(R%))=4 ORA%(E%(R%)
,F%(R%))=5:E%(R%)=K%:F%(R%)=L%:PROChyp:
SOUND0,-15,200,1
 2160 IFE%(R%)<0 E%(R%)=0
 2170 IFE%(R%)>19 E%(R%)=19
 2180 IFA%(E%(R%),F%(R%)+1)=0 F%(R%)=F%
(R%)+1:A%(E%(R%),F%(R%))=6:TRAP%(R%)=TR
UE::T%(R%)=TIME
 2190 IFA%(E%(R%),F%(R%))=0 A%(E%(R%),F
%(R%))=4
 2200 IFA%(E%(R%),F%(R%))=2 A%(E%(R%),F
%(R%))=5
 2210 IFA%(K%,L%)=4 PRINTTAB(K%,L%);SPC
1:A%(K%,L%)=0
 2220 IFA%(K%,L%)=5:COLOUR1:PRINTTAB(K%
,L%);L$:A%(K%,L%)=2
 2230 COLOUR3:PRINTTAB(E%(R%),F%(R%));G$
 2240 IF(X%=E%(R%)ANDY%=F%(R%))OR(B%=E%
(R%)ANDC%=F%(R%))THENPROCdead
 2250 ENDPROC
 2260 :
 2270 DEFPROChyp:REPEAT:E%(R%)=RND(19):
F%(R%)=(RND(9)*3)+1:UNTILE%(R%)<>X% AND
F%(R%)<>Y% ANDE%(R%)<>B% ANDE%(R%)<>C%
ANDA%(E%(R%),F%(R%))<>4 ANDA%(E%(R%),F%
(R%))<>5
 2280 ENDPROC
 2290 :
 2300 DEFPROCstill
 2310 IFK%=19 P%=-1:SG%(R%)=FALSE ELSEI
FK%=0 P%=1:SG%(R%)=FALSE
 2320 IF(A%(K%+P%,L%)=4 ORA%(K%+P%,L%)=
5)SG%(R%)=FALSE:PROChyp ELSEE%(R%)=E%(R
%)+P%
 2330 ENDPROC
 2340 :
 2350 DEFPROCsu
 2360 DEAD%=0
 2370 COLOUR3
 2380  FORI%=1 TO3:E%(I%)=8:NEXTI%
 2390 F%(1)=4:F%(2)=13:F%(3)=25
 2400 FORI%=1TO3
 2410 PRINTTAB(E%(I%),F%(I%));G$:A%(E%(
I%),F%(I%))=4
 2420 NEXTI%
 2430 REPEAT:X%=RND(18):Y%=(RND(8)*3)+1
:UNTILA%(X%,Y%)=0 ANDA%(X%-1,Y%)=0:Z%=1
:B%=X%-1:C%=Y%
 2440 COLOUR3:PRINTTAB(X%,Y%);M$(1):COL
OUR2:PRINTTAB(B%,C%);PRL$(1)
 2450 COLOUR3:PRINTTAB(3,3);"SHEET ";U%
 2460 COLOUR3:FORI%=1TOH%:PRINTTAB(I%,3
1)M$(1);:NEXTI%
 2470 ENDPROC
 2480 :
 2490 DEFPROCdead:PRINTTAB(H%,31);SPC1;
:H%=H%-1
 2500 DEAD%=TRUE
 2510 *FX15,0
```

→

```
 2520 COLOUR3:SOUND0,3,5,1
 2530 PRINTTAB(B%,C%);SPC1
 2540 PRINTTAB(X%,Y%);MD$
 2550 ENDPROC
 2560 :
 2570 DEFPROCfall
 2580 LA%=TRUE
 2590 PROCprint
 2600 COLOUR3:LA%=FALSE
 2610 REPEAT
 2620 COLOUR3:PRINTTAB(X%,Y%);M$(3)
 2630 IFA%(X%,Y%)=6 PROCdead
 2640 IFA%(X%,Y%)=-1 COLOUR2:PRINTTAB(X
%,Y%);P$ ELSEPRINTTAB(X%,Y%);SPC1
 2650 Y%=Y%+1
 2660 UNTILA%(X%,Y%+1)=-1 ORA%(X%,Y%+1)
=7 ORA%(X%,Y%+1)=8 ORDEAD%
 2670 IFDEAD% ENDPROC
 2680 PRINTTAB(X%,Y%);M$(3):B%=X%-1:C%=
Y%-1:Z%=3
 2690 IFA%(X%,Y%)=4 PROCdead
 2700 ENDPROC
 2710 :
 2720 DEFPROCdig
 2730 IFA%(B%,C%+3)=2 ORA%(B%,C%+1)=0 O
RA%(B%,C%)=2 ORY%=28 ORZ%=3 ORB%=0 ORB%
=19 ENDPROC
 2740 COLOUR2:PRINTTAB(B%,C%);PD$(Z%):P
ROCg:IFDEAD% ENDPROC
 2750 COLOUR2
 2760 IFA%(B%,C%+1)=6 PROCmonstfall:GOT
O2810
 2770 SOUND0,1,206,1
 2780 IFA%(B%,C%+1)=-1 A%(B%,C%+1)=7:PR
INTTAB(B%,C%+1);B$(1):GOTO2810
 2790 IFA%(B%,C%+1)=7 A%(B%,C%+1)=8:PRI
NTTAB(B%,C%+1);B$(2):GOTO2810
 2800 IFA%(B%,C%+1)=8 A%(B%,C%+1)=0:PRI
NTTAB(B%,C%+1);SPC1
 2810 COLOUR2:PRINTTAB(B%,C%);PRL$(Z%)
 2820 ENDPROC
 2830 :
 2840 DEFPROCmonstfall
 2850 LV%=1
 2860 FORI%=1TO3
 2870 IFDM%(I%)GOTO2890
 2880 IFE%(I%)=B% ANDF%(I%)=C%+1 XX%=E%
(I%):YY%=F%(I%):RR%=I%
 2890 NEXTI%
 2900 COLOUR2:PRINTTAB(XX%,YY%);P$:A%(X
X%,YY%)=-1
 2910 YY%=YY%+1:COLOUR3:PRINTTAB(XX%,YY
%);G$
 2920 REPEAT
 2930 IFA%(XX%,YY%)=0 PRINTTAB(XX%,YY%)
;SPC1 ELSEIFA%(XX%,YY%)=6 PRINTTAB(XX%,
YY%);G$
```


```
2940 YY%=YY%+1
2950 PRINTTAB(XX%,YY%);G$
 2960 IFINT((YY%-2)/3)=(YY%-2)/3 LV%=LV
%+1
 2970 UNTILA%(XX%,YY%+1)=-1 ORA%(XX%,YY
%+1)=7 ORA%(XX%,YY%+1)=8
 2980 TRAP%(RR%)=FALSE
 2990 IFLV%>=U% PRINTTAB(XX%,YY%);SPL$:
SOUND0,3,5,1:PRINTTAB(XX%,YY%);SPC1:S%=
S%+(LV%*100):DM%(RR%)=TRUE:CO%=CO%+1 EL
SEE%(RR%)=XX%:F%(RR%)=YY%:A%(XX%,YY%)=4
 3000 ENDPROC
 3010 :
 3020 DEFPROCscore
 3030 COLOUR3:PRINTTAB(3,1);"SCORE ";S%
 3040 BON%=OXY%-(INT(TIME/100))
 3050 IFBON%<=99 PRINTTAB(15,31);SPC1;
 3060 IFBON%<=9 PRINTTAB(14,31);SPC1;
 3070 PRINTTAB(6,31);"OXYGEN ";BON%;
 3080 IFBON%<=0 PROCdead
 3090 ENDPROC
 3100 :
 3110 DEFPROCback(BX%,BY%)
 3120 IFA%(BX%,BY%)=0 PRINTTAB(BX%,BY%)
;SPC1:ENDPROC
 3130 IFA%(BX%,BY%)=2 COLOUR1:PRINTTAB(
BX%,BY%);L$:ENDPROC
 3140 IFA%(BX%,BY%)=-1 COLOUR2:PRINTTAB
(BX%,BY%);P$:ENDPROC
 3150 IFA%(BX%,BY%)=7 COLOUR2:PRINTTAB(
BX%,BY%);B$(1):ENDPROC
 3160 IFA%(BX%,BY%)=8 COLOUR2:PRINTTAB(
BX%,BY%);B$(2):ENDPROC
 3170 ENDPROC
 3180 :
 3190 ON ERROR OFF
 3200 MODE 6
 3210 IF ERR=17 END
 3220 REPORT:PRINT" at line ";ERL
 3230 END
```

CASSETTE TROUBLES

By Peter Rochford

This month Peter Rochford takes a close look at the problems that you can encounter when using cassettes for storing programs, and gives some suggestions to help alleviate the troubles that you may come across.

If there is a particular aspect of using a micro that causes more frayed tempers than any other, it must be loading and saving cassette programs. I must admit that, compared to other computers, the Electron like its bigger brother the BBC micro does have a better cassette interface than most.

Still, problems do occur from time to time for various reasons and computer shops seem to spend a great deal of their time sorting out customers' troubles in this area. Remembering to observe a few golden rules and armed with a bit of extra knowledge, most of these problems can be avoided. We'll start by discussing two of the simplest (and in my experience most neglected) causes of problems with loading and saving.

THE CASSETTES

Firstly, the cassettes you use on your deck must be of good quality. Most of the so-called 'computer cassettes' that bear no brand name or label, are of dubious quality. The tape they contain is thin, possessing poor magnetic qualities, has a tendency for the brown oxide coating to flake off, and is very prone to creasing or being chewed up by the cassette deck tape transports. Even the cassette shell and its mechanism are made to poor tolerances, causing uneven tape feed and possibly uneven tape speed.

The Japanese company TDK, well known for their high quality audio cassettes, are now producing C15 tapes suitable for home computer use. These tapes are excellent and the only ones that I would recommend [We have found that any 'quality' brand will normally produce good results - Ed.]. If you are unable to locate a shop selling these, try to obtain the C45 by the same manufacturer that sells for around the same price. This may seem a better buy at first, but the extra time spent winding and rewinding is detrimental to the cassette deck and the tape in the long run, not to mention inconvenient when in a hurry. If you find that 22 minutes per side is too long, you can always cut and splice the tape to the most suitable length. It's worth all the effort I can assure you.

THE CASSETTE RECORDER

Next, the importance of correct care and cleaning of the heads and tape transport of your deck cannot be overstressed. Sadly, so many people seem to neglect this and the results are damaged tapes and constant load and save problems. I blame manufacturers for this, as they never seem to emphasize the importance of good maintenance in the literature they supply with the machine.

The use of 'head-cleaning tapes' is a real waste of time and money and has my total condemnation. I spent ten years in the hi-fi business and never came across a head-cleaning tape that cleaned the heads effectively - let alone the rest of the tape transport. The best way to do the job is with cotton wool buds and a proprietary head-cleaning fluid. A company called BIB market a kit containing all the things you need and this is available from most of the better hi-fi dealers. Instructions are included on cleaning the heads, rubber pinch roller, capstan and tape guides. How often you perform this operation depends on how often you use the deck, but an average guide is every three weeks if the machine is in regular use.

With periodic attention paid to the cleaning of the transport of your tape machine and the use of good quality tape you will save yourself a lot of aggravation. So, now let's look at some

of the more involved problems with cassette machines and their causes.

A great many owners of computers I know have all experienced the annoyance of getting their new computer home and being unable to get the demonstration tape, or other commercially made tape to load with their cassette deck. This may be due to the deck being incompatible with the cassette interface of the Electron. However, quite often it is a case of the record/playback head-alignment of your cassette recorder being different to that of the duplicating machine the demonstration tape was made on.

The problem occurs because when the tape passes over the head on your deck, the position of the recorded signal is mis-aligned on the tape in relation to the position of the head. The mis-alignment is never so far out that the head picks up no signal at all, but just enough to cause a loss in level and information.

The cure is for the position of the head on your machine to be adjusted by means of a screw on the platform that the head sits on. I don't intend to give you instructions on how to do it yourself for two reasons. Firstly, all cassette decks differ in how accessible the screw is, making a description too involved. Secondly, it is all too easy to get into a lot of trouble unless you know what you're doing. I therefore urge you to take your deck to your computer dealer or hi-fi specialist and get his engineer to align it using a commercially recorded engineering test tape.

If you encounter only the occasional load or save problem using your computer, try checking the connections on the leads you are using and if they are even slightly suspect, re-solder them using a fine-tipped iron and a good quality resin-cored solder. Plugs that wobble in their sockets and do not provide a firm connection should obviously be replaced.

Getting a consistently good save or load success rate may be just a matter of experimenting with the tone and volume controls of your recorder until you find the best setting. Once you have discovered the optimum setting, mark the controls accordingly for future reference. On some decks, if you are using a DIN socket, no amount of fiddling with the controls will do anything, as the level of these is fixed and independent of volume and tone settings.

Electrical interference can often be the hidden source of trouble. The switching on and off of the thermostat on your fridge or central heating can send a nasty spike through your mains supply and cause your load or save to fail at some point. The best answer here is, once you have identified the offending unit, have it fitted with a suppressor by a qualified electrician. There are devices on the market which you can plug your computer into to suppress mains borne interference. Sometimes they work, but it may be that the interference is not coming through the mains itself, but transmitted through the air. So it is better to suppress the problem at source.

The last question I am going to look at is the most complex. That of tape deck compatibility with the Electron. You should try and get a machine that features a five pin DIN socket and preferably has remote motor control. Unfortunately, many of the machines on the market feature five pin DIN sockets which have fixed output levels. Provided the specification is within that given elsewhere in this article and the machine is of high quality this probably won't be a problem; there can be no guarantees though. Try and avoid a machine which relies on the earpiece output for loading and the microphone input socket for saving. The load impedances in both cases are wrong and the output of the earpiece socket at high volume settings may be large enough to damage the cassette interface in your computer. The input/output levels and impedances for a tape deck to match the Electron should be as shown in Table 1.

Any decent hi-fi dealer should be able to help you choose the right deck given these figures. Don't however, think they will guarantee the deck will work with the Electron. Bear in mind

Input Sensitivity	- Between 65 to 100mv peak-to-peak
Output Level	- Between 20mv to 5v peak-to-peak
Input Impedance	- Greater than 100k
Output Impedance	- Less than 200ohms

Table 1.

the question of head-alignment we discussed earlier, which may mean taking the deck to your dealer for adjustment. Tape speed can be quite critical on a cassette machine when used with a computer. The maximum variation in speed from the mean value of one-and-seven-eighths inches per second is expressed as the wow and flutter. The wow and flutter figures should be lower than plus or minus 2% on any machine you wish to consider buying.

The really expensive stereo radio/cassette recorders usually allow manual or automatic recording level. Use the manual control if yours has this facility and set the level on the recording meters to just below the zero VU mark. Also, don't forget to switch the unit into mono mode if possible.

Don't be tempted to go for the cheapest machine. The extra expense will mean a deck which has a better chance of performing reliably, and with greater longevity. In addition, if you buy a good machine you can use it for other tasks apart from a data recorder.

Finally, there are a few other do's and don'ts to remember that might help save you some problems.

Apart from keeping the tape heads clean, it is worthwhile buying or borrowing a head demagnetiser. These devices should be used carefully and with strict adherence to the instructions that come with the unit.

Correct storage of tapes is an often overlooked consideration. Don't leave them near magnetic fields such as those produced by loudspeakers, electric motors, mains transformers and TV monitors. Always return them to their protective cases when not in use and whatever you do, don't touch the surface of the tape itself.

If your machine has remote motor control, never leave it in the play mode with the motor off for longer than is necessary. Whilst in this situation, the metal capstan is in contact with the soft rubber pinch roller causing it to deform and this will subsequently lead to irregular tape feed. It will also put a crease in a tape that is in situ, leading to signal dropout when that portion is next used.

We all curse cassette loading at some time or other for various reasons. Amongst its many disadvantages are that of speed and flexibility. If you follow the above advice, I am sure it will help to remove some of the pain for you.

HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS

FASTER "AND" IN IF STATEMENTS - S. Williams

When using multiple conditions in an IF statement it is marginally faster to use nested IF commands to replace the AND operator. For example:

```
IF A>0 THEN IF B>0 THEN PROCdo
```

is marginally faster than

```
IF A>0 AND B>0 THEN PROCdo
```

DIRECT POKE WITH INPUT - J.S. Wellsman

It is possible to poke a value directly into a memory address with the INPUT command, for example: INPUT "A value "?&70 will input a value and store it at memory address &70. The value must be expressed in decimal and, of course, cannot be greater than 255.

ELECTRON GRAPHICS (Part 10)

by Mike Williams

This month, in the last part of this series on Electron graphics, we show you how to achieve really fast and smooth animation of graphics displays. We have also tried to include several interesting programs. These illustrate the techniques involved and serve to round off the series.

The instructions needed to produce animation have already been discussed in some detail in the previous articles in this series. The animation technique itself depends heavily on the use of VDU19, which we introduced in part 8. So to make a start, let's recap on this useful and versatile feature.

The VDU19 instruction essentially requires two parameters, one to refer to a colour relative to the current graphics mode, and the other to refer to the actual colour to be used. In addition, there must be a further three bytes, each zero, just for completeness.

For example, in modes 1 and 5, we are able to use four colours referred to by number as 0, 1, 2, and 3. These are the relative colour values (RCV) and they can be changed to any of the 16 actual colours available on the Electron by using the VDU19 instruction. By default, colours 0, 1, 2, and 3 are black, red, yellow and white in modes 1 and 5. However, if we execute the following two instructions:

```
VDU19,0,4,0,0,0
VDU19,3,2,0,0,0
```

then the same four numbers will refer to the colours blue, red, yellow and green, because we have changed the relative colours 0 and 3.

Thus in any graphics mode, whether two, four or sixteen colours are in use we can always use the VDU19 instruction to turn these into any set of colours we like out of the total of 16 (including flashing colours) that are possible. Indeed we could select mode 2 and change all 16 relative colours so that they all appear as black (or any other colour we care to choose) but we would then be drawing in black on a black background and nothing would be visible. However, although this doesn't seem initially very useful, this is the whole basis of the animation technique that we are going to use.

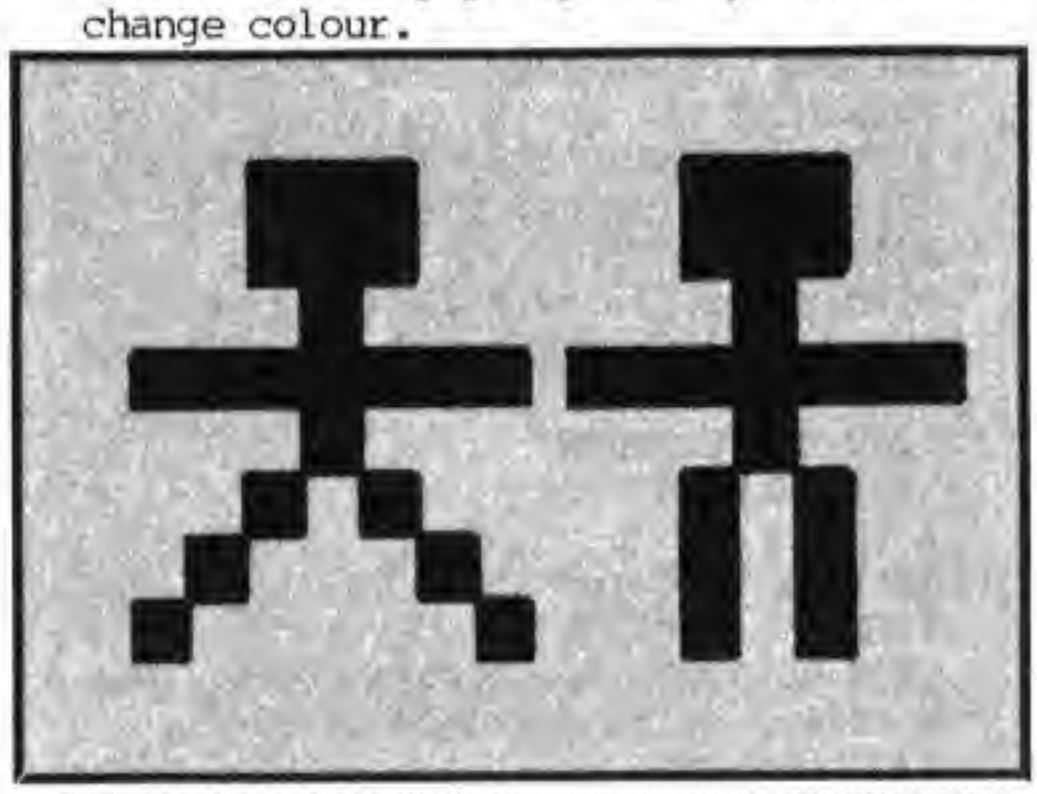
Suppose we select mode 2 (this gives us the use of the maximum 16 colours) and we then change all of these to black using VDU19. Next we will program the computer to draw some object in 15 different positions on the screen, each time using a different relative colour (1, 2, 3 etc). If we now use VDU19 to change first colour 1 to white, then colour 2 to white and so on, the object will seem to appear in each of the positions where it was originally drawn. With the right set of positions this will make the object appear to move, and very quickly indeed. We shall always need a background, so we will leave relative colour 0 (black) alone and just manipulate the other 15.

Let's now have a look at an example. The program ROTATE when run appears to make a square rotate about a central point on the screen. The main program is quite short - VDU29 resets the graphics origin to the centre of the screen, PROCblackall sets all 15 'colours' in mode 2 to black, and PROCsquares(200) draws 15 squares of size 200 evenly rotated about the central point and all in black so that they cannot be seen at the start. Then

```
 10 REM Program ROTATE
 20 REM Version E1.1
 30 REM Author Mike Williams
 40 REM ELBUG October 1984
 50 REM Program subject to copyright
 60 :
100 MODE 2
110 VDU23,1,0;0;0;0;
120 ON ERROR GOTO 190
130 VDU29,640;512;
140 PROCblackall
150 PROCsquares(200)
160 PROCrotate
170 END
180 :
190 ON ERROR OFF:MODE 6
200 IF ERR=17 THEN END
210 REPORT:PRINT" at line ";ERL
220 END
230 :
1000 DEF PROCblackall
1010 FOR X%=1 TO 15
1020 VDU19,X%,0,0,0,0
1030 NEXT X%
1040 ENDPROC
1050 :
1060 DEF PROCsquares(size)
1070 FOR C%=1 TO 15
1080 GCOL0,C%:angle=2*PI*C%/15
1090 PROCdrawsquare(size,angle)
1100 NEXT C%
1110 ENDPROC
1120 :
1130 DEF PROCdrawsquare(size,angle)
1140 x1=size*COS(angle):y1=size*SIN(an
gle)
1150 MOVE0,0:DRAW x1,y1
1160 DRAW x1-y1,x1+y1
1170 DRAW -y1,x1:DRAW 0,0
1180 ENDPROC
1190 :
1200 DEF PROCrotate
1210 REPEAT
1220 FOR C%=1 TO 15
1230 VDU19,C%-1,0,0,0,0
1240 VDU19,C%,7,0,0,0
1250 T%=INKEY(5)
1260 NEXT C%
1270 VDU19,15,0,0,0,0
1280 UNTIL FALSE
1290 ENDPROC
```

PROCrotate simply changes each of the colours 1 to 15 in turn to white. This gives the effect of a square rotating about the centre of the screen and so fast that the delay at line 1250 is needed to slow things down a bit. The beauty of this method is that the apparent speed of animation is quite independent of the complexity of the object being animated since it is only the colour which is being changed, nothing is being erased or redrawn.

The program ROTATE shows this technique being used with line drawing graphics (MOVE, DRAW, and PLOT). Similar effects can be achieved with user defined characters. In the program called WALK a small man is defined in two positions (legs together, legs astride) as characters 240 and 241 at the start of the program. Again using mode 2 all the 15 colours are set to black and the man drawn alternately in his two versions across the screen (PROCsetup). Once again by changing the colour in each position in turn to white and back to black, we can make the man appear to walk across the screen. In this case, two loops in the procedure PROCwalk first make him walk from left to right, and then back again from right to left. All this is achieved simply by using VDU19 to change colour.



```
 10 REM Program WALK
 20 REM Version E1.1
 30 REM Author John Wellsman
 40 REM ELBUG OCTOBER 1984
 50 REM Program subject to copyright
 60 :
100 MODE 6
110 ON ERROR GOTO 220
120 INPUTTAB(5,10)"Delay (in centisec
onds)? "T
130 MODE 2:VDU23,1,0;0;0;0;
140 VDU23,240,28,28,8,127,8,20,34,65
150 VDU23,241,28,28,8,127,8,20,20,20
160 :
170 PROCblackall
```

```
 180 PROCsetup
 190 PROCwalk
 200 END
 210 :
 220 ON ERROR OFF:MODE 6
 230 REPORT:PRINT" at line ";ERL
 240 END
 250 :
1000 DEFPROCsetup
1010 FOR X%=0 TO 15
1020 COLOUR X%
1030 PRINTTAB(X%,10)CHR$(240+X% MOD 2)
:REM Each successive character printed
to each of the fifteen relative colour
values in this mode.
1040 NEXT X%
1050 ENDPROC
1060 :
1070 DEFPROCwalk
1080 REPEAT
1090 FOR X%=0 TO 15
1100 C%=X%:IF C%=0 THEN C%=1
1110 M=INKEY(T)
1120 VDU19,C%-1,0,0,0,0:REM This line
blacks out the previous character displ
ayed.
1130 VDU19,C%,7,0,0,0:REM This changes
 the relative colour value of the next
character to white.
1140 NEXT X%
1150 FOR X%=15 TO 0 STEP -1
1160 C%=X%:IF C%=0 THEN C%=1
1170 M=INKEY(T)
1180 VDU19,C%+1,0,0,0,0
1190 VDU19,C%,7,0,0,0,0
1200 NEXT X%
1210 UNTIL FALSE
1220 ENDPROC
1230 :
1240 DEFPROCblackall
1250 FOR X%=1 TO 15
1260 VDU19,X%,0,0,0,0:REM Every relati
ve colour value switched to black.
1270 NEXT X%
1280 ENDPROC
```

Of course there are some limitations. We are really forced to use mode 2 to provide sufficient colours and hence positions for an object. Because of the need for a background colour we are already restricted to 15 positions, and if we wish to display anything else on the screen at the same time then this will use up more colours, further restricting the number that we can use for animation (otherwise any text or other graphics would also appear to switch on and off). Nevertheless the technique is a very useful one indeed and widely used.

Let's have a look at some further variations on this basic idea. The next example called RIPPLE produces a different kind of movement. The program again uses mode 2 and sets colours 1, 2, and 3 to be blue, cyan and white. The program then draws a series of short horizontal lines diagonally from the bottom to the top of the screen using each of the three colours in rotation. The remainder of the program

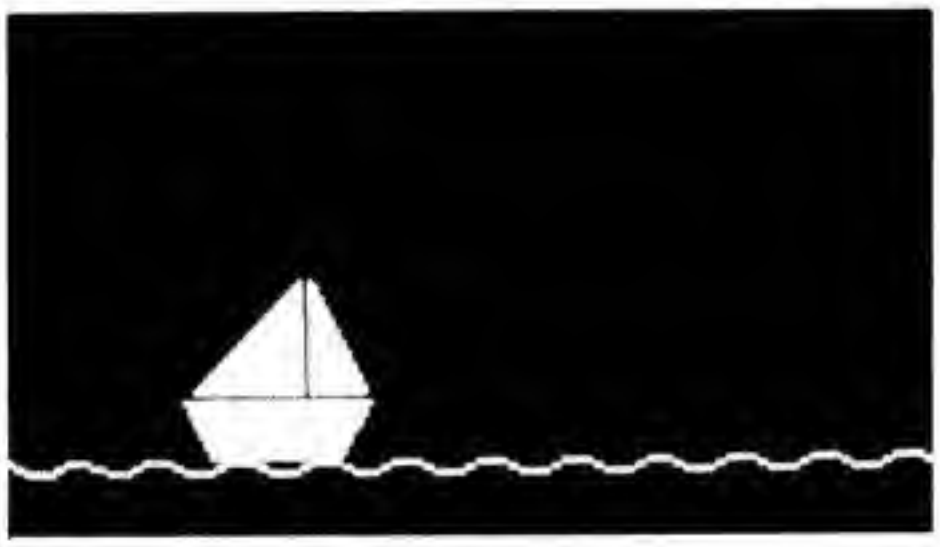
```
  10 REM Program RIPPLE
  20 REM Version E1.2
  30 REM Author : John Wellsman
  40 REM ELBUG OCTOBER 1984
  50 REM Program subject to copyright
  60 :
 100 ON ERROR GOTO 410
 110 MODE 2
 120 VDU23,1,0;0;0;0;
 130 colour=1
 140 REM This sets RCV 1,2 & 3 initial
ly to blue, cyan & white
 150 VDU19,1,4,0,0,0
 160 VDU19,2,6,0,0,0
 170 VDU19,3,7,0,0,0
 180 FOR Y%=0 TO 1000 STEP 10:REM This
 FOR loop draws horizontal lines
 190 GCOL0,colour:REM This draws each
line in RCV 1,2 & 3 successively
 200 MOVE 100+Y%/1.5,Y%:DRAW 500+Y%/1.
5,Y%
 210 colour=colour+1:IF colour=4 colou
r=1:REM Increases RCV
 220 NEXT Y%
 230 REPEAT
 240 VDU19,1,7,0,0,0
 250 VDU19,2,4,0,0,0
 260 VDU19,3,6,0,0,0
 270 M%=INKEY(10)
 280 VDU19,1,6,0,0,0
 290 VDU19,2,7,0,0,0
 300 VDU19,3,4,0,0,0
 310 M%=INKEY(10)
 320 VDU19,1,4,0,0,0
 330 VDU19,2,6,0,0,0
 340 VDU19,3,7,0,0,0
 350 M%=INKEY(10)
 360 UNTIL FALSE
 370 REM Lines 240 to 340 progressivel
y change RCV 1,2 & 3 through blue, cyan
 and white.
 380 REM Lines 270,310 & 350 induce a
delay.
```

```
  390 END
  410 MODE6:ON ERROR OFF
  420 IF ERR<>17 THEN REPORT:PRINT" at
line ";ERL
  430 END
```

then progressively changes each of the relative colours 1, 2, and 3 through the cycle of three colours (7, 4 and 6). Three complete changes return the picture to its original state. The effect on the screen is of the three colours rippling up the screen. Again the apparent movement is completely illusory, being purely the result of carefully managed colour changes.

We have already seen that the technique is limited by the number of colours, thus we can place an object in just 15 different positions on the screen. However, we can still apply a variation on the basic theme when many more positions are being used. The advantage of having a large number of positions is that the movement appears to be much smoother. The problem when using the MOVE, DRAW and PLOT instructions is that you see the object as it is being drawn on the screen. If it is then erased, maybe by using the Exclusive-OR technique described much earlier in this series, a jerky and flickering effect results.

This is illustrated in the next example called SHIP1 - its purpose is to make a sailing ship move across the sea from left to right. The procedure PROCsail draws the ship, (made up of triangles) by calling PROCdraw first to display the ship, and then to make it disappear by drawing it a second time in the background colour. This is done progressively across the screen.



```
  10 REM Program SHIP1
  20 REM Version E1.2
  30 REM Author Colin Opie
  40 REM ELBUG OCTOBER 1984
  50 REM Program subject to copyright
  60 :
 100 MODE 1
 110 VDU23,1,0;0;0;0;
 120 ON ERROR GOTO 190
 130 VDU19,3,6,0,0,0
 140 GCOL0,3:PROCsea
 150 PROCsail
 160 VDU19,3,7,0,0,0
 170 END
 180 :
 190 ON ERROR OFF:MODE 6
 200 REPORT:PRINT" at line ";ERL
 210 END
 220 :
1000 DEF PROCsea
1010 FOR X=0 TO 1279 STEP 4
1020 Y=50+5*SIN(X/10)
1030 PLOT69,X,Y
1040 NEXT X
1050 ENDPROC
1060 :
1070 DEF PROCdraw(X%,Y%,C%)
1080 PLOT4,X%,Y%
1090 PLOT4,X%+80,Y%
1100 PLOT80+C%,X%+80,Y%+80
1110 PLOT4,X%+88,Y%
1120 PLOT4,X%+88,Y%+80
1130 PLOT80+C%,X%+128,Y%
1140 PLOT4,X%,Y%-8
1150 PLOT4,X%+128,Y%-8
1160 PLOT80+C%,X%+20,Y%-48
1170 PLOT4,X%+24,Y%-48
1180 PLOT4,X%+132,Y%-8
1190 PLOT80+C%,X%+108,Y%-48
1200 ENDPROC
1210 :
1220 DEF PROCsail
1230 GCOL0,2
1240 FORX=0 TO 1279 STEP 4
1250 PROCdraw(X,105,5)
1260 PROCdraw(X,105,7)
1270 NEXT X
1280 ENDPROC
```

Unfortunately, the result is not very satisfactory for the reasons described above. The solution is each time to draw the ship in the next position in black while still visible in the previous position, and then by changing colour to switch off the first image and switch on the second. Because we no longer see the ship being drawn the resulting movement is very much

smoother. We can change our program SHIP1 into the new version (SHIP2) by replacing the procedure PROCsail by the new version below together with the additional procedure PROCshow.

```
1220 DEF PROCsail
1230 VDU19,3,3;0;
1240 FOR X=0 TO 1279 STEP 16
1250 GCOL1,1
1260 PROCdraw(X,105,5)
1270 PROCshow(1)
1280 GCOL1,2
1290 PROCdraw(X+8,105,5)
1300 PROCshow(2)
1310 GCOL3,1
1320 PROCdraw(X,105,5)
1330 GCOL1,1
1340 PROCdraw(X+16,105,5)
1350 PROCshow(1)
1360 GCOL3,2
1370 PROCdraw(X+8,105,5)
1380 NEXT X
1390 ENDPROC
1400 :
1410 DEF PROCshow(S%)
1420 P%=3:REM YELLOW
1430  VDU19,S%,P%,0,0,0
1440  VDU19,3-S%,0,0,0,0
1450 ENDPROC
```

The new version is longer because there is a complication in this new technique. Each of the new ships overlaps the previous one, so we actually need to use three relative colours. Colour 1 is used for the first display and colour 2 for the second. On the third display we want to erase the display drawn in colour 1, but not the one in colour 2. This is achieved by using GCOL1,c when drawing the ship and GCOL3,c when erasing the ship where c is the relative colour, alternately 1 and 2. Relative colour 3 is set to yellow, the colour of the ship, at the start of the procedure PROCsail, because any overlapping points between two images are always visible until the previous image is erased. The purpose of the procedure PROCshow is to switch colours 1 and 2 between black and yellow each time it is called, causing one image to appear and the other to disappear, ready to be erased before the next image is invisibly drawn.

Clearly, from this last example, animated graphics can quickly become a complicated subject. If you want to understand the techniques described here fully, then time spent studying and playing with these programs will be well rewarded.

This now completes our series on Electron graphics. We have progressed from very basic ideas to some quite sophisticated and complicated techniques. The world of graphics is a fascinating one and also a most rewarding one for the computer programmer. Have fun.

HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS

REVERSING FLAGS - R. Sterry

To change a boolean flag between its states (FALSE and TRUE) you could use:

```
IF FLAG%=TRUE THEN FLAG%=FALSE ELSE FLAG%=TRUE
```

More elegant however is:

```
FLAG%=NOT FLAG%
```

SIMULATED BBC TAB KEY - Edward Westhead

A lot of BBC programs will work on the Electron, but some use the Tab key, which is not present. For most programs (apart from some games), this can be simulated by pressing the Ctrl key down and at the same time pressing the 'I' key, still with the Ctrl key held down. This is known as a Control-I.

WAITING FOR KEYS - K. Allen

To make a program wait until no keys are being pressed use:

```
REPEAT UNTIL INKEY(-129)
```

To make it wait for a key to be pressed, use:

```
REPEAT UNTIL NOT INKEY(-129)
```

NEW GAMES FOR YOUR ELECTRON

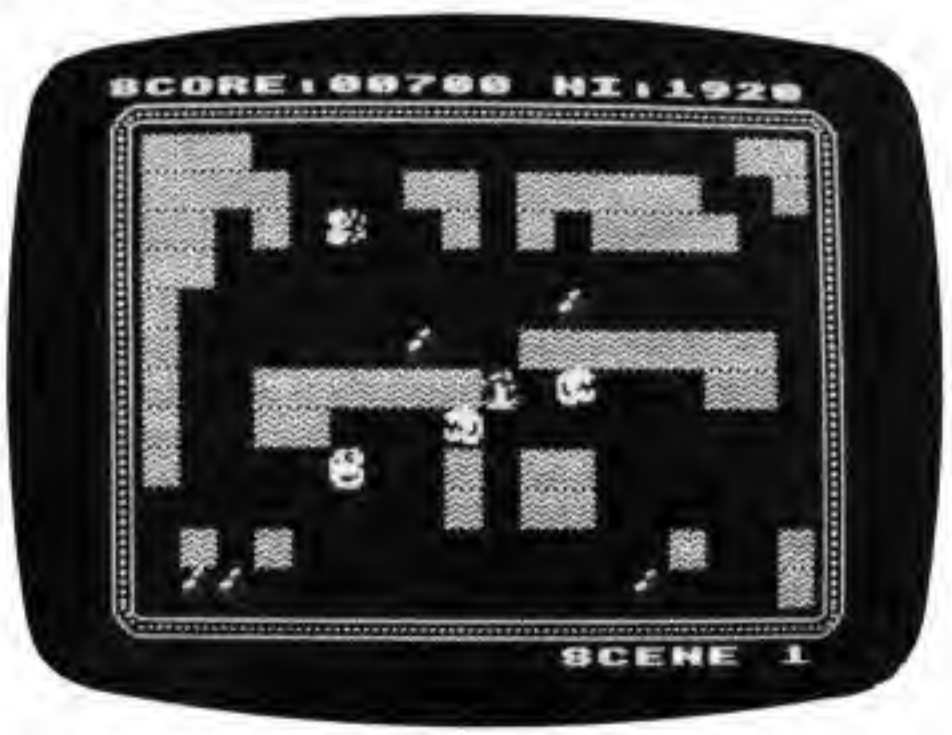
Name : Ghouls
Supplier : Micro Power
Price : £7.95
Reviewer : Alan Webster
Rating : ****

'Ghouls' takes place inside a creepy mansion set on the top of a hill. Your aim is to find the "power jewels" whilst trying to avoid the nasty inhabitants of the mansion, deadliest of which is the ghost. You have to run and jump around the floors, avoiding poisonous spiked traps and trying not to fall off the escalators. Avoid the ghosts at all costs, though these can be made to disappear for a time by eating a stray 'power pill'.

There are four different screens in this game and each screen has its own room name. In my opinion the sound in this game detracts from the overall enjoyment, but this can be switched off if required from within the program.

I am sure that this game will sell well as it is a lot of fun to play. It requires the use of only three keys so that it is easy to operate. As an extra bonus, for people who have a joystick interface, this game will allow you to use switched-type joysticks (i.e the type used by the First Byte interface reviewed in ELBUG Vol.1 No.7) to play the game.

Name : Mr.Wiz
Supplier : Superior Software
Price : £7.95
Reviewer : Alan Webster
Rating : ***



Mr.Wiz is a version of the arcade game 'Mr.Do' in which you take the role of a wizard trying to collect the cherries in his garden. There are the usual gremlins to avoid, and these can be killed by dropping apples on them. You also have a crystal ball which you can throw at the gremlins to kill them.

There are about eight different sheets, and the game can become quite difficult at some stages. Despite this, my overall impression is that the movement and sound in this game are disappointing. The basic concept on which the game is based is certainly a good one, but I did not find that the action was fast enough in this implementation to produce an enjoyable game playing experience.

Name : The Wheel of Fortune
Supplier : Epic Adventures
Price : £9.95
Reviewer : John Waller
Rating : ****

Adventure games have something of a cult following amongst micro enthusiasts. This excellent adventure from Epic contains some of the advanced

features first seen in 'The Hobbit' from Melbourne House. The aim in this adventure is to find the 'Wheel of Fortune' and escape the mystical world into which you have been plunged. There are some 'intelligent' characters in the game who help (or hinder) your progress. Be careful how you treat these as they have varying moods, and will do their best to kill you or lock you up in jail if you ill treat them. You have been warned!

Complex commands of up to 254 characters are allowed. You can talk to the more intelligent characters, but they tend to be a bit deaf and not very helpful. The function keys can be defined to store the more common instructions to speed your progress through the game. There is also a very fast 'Save game' facility, so you don't have to work your way through the initial stages of the game every time you play.

Overall this is an excellent adventure, which will prove satisfying to the novice adventurer, yet still be challenging to the more experienced player.

MULTIPLE PROGRAMS IN MEMORY

By Nigel Harris

If you've had your Electron for many months now, you're perhaps already wondering if there are ways of reducing the number of times that you have to load a program from your cassette recorder. Here is a simple idea which will help do just that. It also allows one program to operate on another in memory at the same time.

In the Acorn Electron you have at your mercy 32k of "random access memory". Of course not all of this is directly available for your use. Some is used by the computer's operating system and some by the computer's peripheral ports; the rest is left for you and your program. When using Basic, you can expect to have as much as 20k of space for a program, though this will depend on the screen mode chosen. If you're like me, the chances are that the great majority of your programs use only a part of this, the rest going vacant for the duration of the program. When one realises this, the obvious question is how the remaining empty space might best be used - how about, for example, having more than one program in memory at the same time?

PAGE, TOP, LOMEM & HIMEM

When a Basic program is loaded in memory, the computer's Basic interpreter needs to know where to find it. That is, where the program starts and ends. It also needs to know where the beginning of unused memory is and where the display memory starts. All of these positions have got names and you can print out their values on the screen.

Typing PRINT ~PAGE (followed by Return - the squiggle is tilde; Ctrl and the left arrow keys together) will print the start address of your current program and it will normally be the value E00, in hexadecimal notation, or rather &E00 in 'computer-speak'. TOP points, as it says, to the top of (or the end of) your program. So after you've loaded a program, printing the difference, TOP-PAGE, will tell you its length in bytes. LOMEM and HIMEM define the empty memory that's left over between the top of the Basic program (LOMEM) and the bottom of the memory that's used for the screen display.

TOP and LOMEM will usually have the

same value before a program is run, but during run time, LOMEM moves away from TOP as the memory between them is used up by the variables in the program. So LOMEM points at the last byte used in this work space or 'heap'.

You can control the values of these variables to some extent but the most useful variable that you can control is PAGE. Provided that you leave enough space for each program and for its heap to grow (this includes space for variables, arrays, procedure and function calls and - most thirstily of all - strings) then you can set PAGE at different levels throughout memory and load a different program at each point.

If you type in a program, don't forget to type in NEW before you start, this forces Basic to reset the other pointers to agree with PAGE's new value. If you load a program this is not necessary as the computer automatically does a NEW itself. Suppose you had 3 short programs "A", "B", and "C" on tape. You could type (assuming that PAGE is already set to &E00, the default setting on switch-on):

```
LOAD "A" <return>
PAGE=&1000 <return>
LOAD "B" <return>
PAGE=&2000 <return>
LOAD "C" <return>
```

After each LOAD command, you should wait while the program is loaded before proceeding to the next step.

To use any one of these three programs, you must first set the pointers to find the one required. All that is needed to do this is to type

```
PAGE=&1000 <Return>
OLD <Return>
```

RUN <Return>

which runs the program "B". For one of the others simply set PAGE accordingly and type OLD and RUN.

You can also arrange for one program in memory to switch automatically to a different program.

As an experiment you might like to try the following two programs. Type NEW and then enter these lines (without running them).

```
10 ON ERROR END
20 PRINT TAB(5,5)"PROGRAM ONE here"
30 TIME=0:REPEAT UNTIL TIME=200
40 MODE 6
50 PAGE=PAGE+&100:RUN
```

Now change the value of PAGE by typing

```
PAGE=PAGE+&100 <Return>
NEW <Return>
```

and enter the next short program

```
10 ON ERROR PAGE=PAGE-&100:END
20 PRINT TAB(30,20)"PROGRAM TWO here"
30 TIME=0:REPEAT UNTIL TIME=200
40 MODE 6
50 PAGE=PAGE-&100:RUN
```

Run either program and you will actually find that both will keep running alternately until Escape is pressed. Notice however, that only one program is executed at a time, although removing line 30 from both programs may make it look as though they were running simultaneously. An enlargement on this technique is used in bigger computers to make them appear to do many jobs at the same time.

This is rather a brief description, but hopefully one that will help in understanding the way that your Basic programs fit into the Electron's memory. Exploiting the ideas touched upon here, you may now find more effective ways of using your machine.

HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS

ANOTHER ROUNDING ERROR - G.Shally

This should return 1 but it actually returns 0:

```
PRINT (40.21*100) MOD 10
```

You can investigate this further with the following two short lines:

```
PRINT INT(40.21*100)        PRINT 40.21*100-4021
```

SPEED IMPROVEMENT WHEN HANDLING LOGICAL VALUES - R. Jefferyes

When storing the logical values TRUE and FALSE these are represented as the numbers -1 and 0 respectively. These can be stored in a floating point variable, but this is much slower than using an integer variable for the same purpose e.g. ok%=TRUE.

COMPACTING BASIC PROGRAMS

by David Tall

If you have problems in getting your larger programs to fit into the available memory, then the utility described here will be a great help in finding the extra space which you require.

INTRODUCTION

Several of the articles that we have published in various issues of ELBUG have emphasized the desirability of presenting programs in a clear and readable format. Indeed this is something we try to achieve with all the programs listed in ELBUG magazine. Clarity and readability are achieved principally by two devices, the inclusion of comments wherever appropriate and the frequent inclusion of spaces to separate the different parts of an instruction.

Of course, excessive use of both these features would considerably lengthen any program and increase the amount of typing when entering a program through the keyboard. Even so it is true to say of most programs that a significant proportion of memory space is used to store information (spaces and comments) which is totally unnecessary for the correct execution of the program by the computer. For example, we normally list all programs in the magazine with one space between the line number and the start of the instruction on each line. There is no need for this at all as far as the computer is concerned, but it certainly helps to make the program more readable.

Now most of the time, the extra memory used up in this way is of no consequence, but sometimes, particularly with larger programs, you can run out of memory, either when the program is running, or even earlier when trying to type the program in. This is most likely to occur when using modes 0, 1 and 2, which require 20K of memory for the screen display, than it is for the other modes where the screen display uses only 10K of memory, for example, in modes 4 and 5.

Sometimes the situation becomes very critical, and it is then that the ability to recover as much free memory space as possible can be invaluable, and the way to do this of course is to remove all unnecessary spaces and comments. Now it would be possible but very tedious to do this by hand, and that is where the utility listed here will come into its own by doing this for you quite automatically.

You may wonder how, if you don't have enough memory for your own program, you can now run a second program to compact and squeeze out the spaces. This is achieved by writing the compacter program in machine code, and storing it lower down in memory in an area not used for Basic programs.

SETTING UP THE PROGRAM COMPACTER

The program listed here, called PACK, should be typed in and saved to cassette as for a normal Basic program. For example:

SAVE "PACK" <return>

Take care if you are unfamiliar with machine code to ensure that you copy the program accurately. You can also save yourself quite a lot of time and effort by omitting the '\' character and any text following this character up to the end of that line. These are just comments and can thus be safely left out.

To produce a working version of the PACK program, run the program and when it stops (the machine code has now been assembled) type as follows:

*SAVE COMPACT B00 D00 B73 <return>

This will save the actual machine code on cassette under the name of COMPACT. This is the program you will use each time you want to perform a compaction.

USING THE COMPACT PROGRAM

There are two ways of running this version of the program. First, however, load in the program to be compacted. Then you can type:

*LOAD COMPACT <return>

in order to load the compacter into memory, and you can then use function

key 1 (Func/f1) to compact your program. Alternatively after loading the program to be compacted you could simply type:

*RUN COMPACT <return>

which will automatically load the machine code and then run it. The compacter occupies that part of memory reserved for function key definitions and user defined characters. Provided you do nothing to alter the contents of these memory areas, the compacter will remain in memory, and may be called by using Func/f1 whenever required.

The compacter program when run offers three independent choices as follows:

SPACEs?	To remove spaces.
REMs?	To remove Basic REM statements.
COMs?	To remove assembler comments.

You should answer Y (yes) or N (no) to each choice in turn. If you don't have any assembler code in your program it will not matter what reply you give to the third choice. If you choose to remove spaces, then only those spaces not essential to the program will be removed. For example, any spaces enclosed within quotes, such as may occur with a PRINT instruction, will always be left alone.

It is quite a good idea with large programs to keep two copies, one containing spaces and comments for readability as the prime copy, and a compacted version as the working copy.

```
 10 REM PROGRAM PACK
 20 REM VERSION E0.1
 30 REM AUTHOR  David Tall
 40 REM ELBUG   OCTOBER 1984
 50 REM PROGRAM SUBJECT TO COPYRIGHT
 60 :
 70 REM To reside in pages &B00, &C00
 80 REM RUN the program & then
 90 REM *SAVE "COMPACT" B00 D00 B73
100 REM The *SAVEd program may be *RUN
110 REM to compact BASIC programs.
120 REM Reply Y to SPACEs? to remove
130 REM redundant spaces.
140 REM Reply Y to REMs? to remove RE
Ms.
150 REM Reply Y to COMMENTs? to delete
160 REM comments in assembler coding.
170 :
180 *FX18
190 *K.1 CA.&B73|L|M
200 :
210 $&B5F="SPACEs?REMs?COMs?"
220 FORN%=0TO1:P%=&B73
230 [OPT3*N%:CLD:LDA#&5E:STA&70:LDA#&
B:STA&71:LDX#2:LDY#0
240 .a LDA(&70),Y:JSR&FFE3:INY:CMP#&3
F:BNEa\print options
250 .x JSR&FFE0:AND#223:CMP#89:BEQy:C
MP#78:BNEx\input response Y or N
260 .y STA&74,X:JSR&FFE3:JSR&FFE7:DEX
:BPLa\store response (& print it)
270    INX:STX&70:STX&72:STX&7C:STX&7
D:STX&80:LDX&18:STX&71:STX&73\    store
 line-start pointers & set ASSEMBLER fl
ag = 0
280 .b CLC:LDA&72:ADC&7D:STA&72:BCCc:
INC&73\start of next input line
290 .c CLC:LDA&70:ADC&7C:STA&70:BCCd:
INC&71\start of next output line
300 .d LDA#0:LDY#4
310 .e STA&77,Y:DEY:BPLe\set all flag
s to zero (except ASSEMBLER)
320 .f INY:LDA(&70),Y:STA(&72),Y:CPY#
3:BEQg:CPY#1:BNEf:CMP#&FF:BNEf:RTS\tran
sfer initial bytes & check if last line
330 .g STA&7C:STA&7D:INY:STY&7E\store
 line lengths & output pointer
340 .h STY&81:LDA(&70),Y:STA&7F\(STAR
T TRANSFER LOOP)      store input point
er & current byte
350    LDX&7B:BNEt:CMP#&8D:BNEt:LDX#4
\check for coded numbers (outside quote
s)
360 .G INC&81:JSRP:DEX:BPLG:BMIh\if c
oded, transfer
370 .t LDX&80:BEQi\if outside ASSEMBL
ER, go to i
380    LDX&74:CPX#&59:BNEu:CMP#&5C:BN
Eu:STX&79\(in ASSEMBLER) check \ & set
COMMENT flag accordingly
390 .u CMP#58:BNEv:LDX#0:STX&79\(in A
SSEMBLER) seek colon and turn off COMME
NT flag if found
400 .v LDX&79:BNEj:CMP#93:BNEj:LDX#0:
STX&80\(in ASSEMBLER)  if outside COMME
NT, seek ] and turn off ASSEMBLER flag
as appropriate; in all cases go to j
410 .i LDX&77:BNEo\(outside ASSEMBLER
) if in REM delete, move on
420    CMP#&22:BNEj:LDA&7B:EOR#1:STA&
7B:LDA&7F\look for ", change QUOTE flag
 as necessary
430 .j LDX&7B:BNEp\if inside QUOTES m
ove on to transfer
440    CMP#91:BNEk:LDX#1:STX&80\(outs
ide QUOTES from here on) if [, set ASSE
MBLER flag
450 .k CMP#&DC:BEQn\DATA?
```

```
  460 .m CMP#&F4:BNEo:LDX&75:CPX#&59:BN
En:LDX#1:STX&77\REM? - if found & delet
ion required, set REM flag
  470 .n LDX#1:STX&7A\set DATA flag (fo
r DATA or REM)
  480 .o LDA&77:ORA&79:BNEq\if REM or C
OMMENT don't transfer
  490    LDA&7A:ORA&80:BNEp:LDA&7F:CMP#
32:BNEp\if DATA  or ASSEMBLER  or not a
 SPACE, do transfer
  500    LDX&76:CPX#&59:BNEp:LDX#0:STX&
82:INY:LDA(&70),Y:JSRsearch:BEQq:DEX:TX
A:EOR#1:STA&82\if SPACES are to be dele
ted, consider following byte
  510    LDY&7E:DEY:LDA(&72),Y:JSRsearc
h:BEQq:LDX#1:INY\look at previous byte
transferred
  520 .A DEX:TXA:ORA&82:STA&82:DEY:LDA(
&72),Y:JSRsearch:BNEA:LDX&82:CPX#0:BEQq
\search earlier bytes
  530 .p LDY&81:JSRP:BNEz\transfer byte
  540 .q LDY&81:DEC&7D\don't transfer
  550 .z CPY&7C:BCCs\check for end of l
ine and repeat as necessary
  560    LDA&79:BEQw:INC&7D\adjust for
COMMENT (1 deletion too many!)
  570 .w LDA&7D:CMP#5:BCCr\if line leng
th less than 5, abort current line tran
sfer
  580    LDY#3:STA(&72),Y:JMPb\else tra
nsfer adjusted line length & move to ne
xt line
  590 .r JMPc\(abort)
  600 .s INY:JMPh\(next byte)
  610 .search LDX#0:CPY#5:BCCF:CMP#&30:
BCCF:CMP#&3A:BCCN:CMP#&40:BCCF:CMP#&5B:
BCCL:CMP#&5F:BCCF:CMP#&7B:BCSF
  620 .L INX
  630 .N INX
  640 .F LDA#&20:CPX#0:RTS\consider byt
e, X = 1 (number), = 2 (letter),  = 0 (
otherwise)
  650 .P LDA(&70),Y:LDY&7E:STA(&72),Y:I
NC&7E:LDY&81:RTS\(transfer):]
  660 NEXT:END
```

TECHNICAL NOTES

It is beyond the scope of this article to explain in detail the working of the compacter program. However, it is interesting and informative to examine some of the ideas involved. In principle, the compacter program needs to start at the first byte or character of the program to be compacted, working through it character by character to analyse the structure and take action as required. Thus the prime need is for an understanding in some detail of how a Basic program is stored in the Electron's memory.

The starting point for any Basic program is referred to as PAGE, and if you print out the value of PAGE (PRINT PAGE <return>) you will see where this is. Unless you have changed the value of PAGE yourself (or a program has done this - see the article in this issue on multiple programs in memory) then this should be the address 3584 (&E00 in hex). A Basic program is stored as a series of coded numbers and we need to be able to examine any memory location to see the number stored there. This is done using a so called 'indirection operator', in this case '?' (see the article in this issue on making programs go faster for more information about indirection operators).

If you type

PRINT ?3584 <return>

this will display on the screen the number stored at memory location 3584. If you try this you should find that the number 13 is displayed as it is a convention on the Electron that the first byte of any Basic program is 13. In fact this is the ASCII code for Return. ASCII codes are used to code numerically all the different characters that are used with your Electron and this includes non-printing characters like Return and Escape.

If you have just switched on your Electron (type NEW <return> if you have been using it for something else) and you display the contents of the next memory location (3585) on the screen in the same way, then you should get the result 255. This value is always used to mark the end of any Basic program in memory. Since there is nothing between the two values of 13 and 255 there is no Basic program at the moment in your Electron.

If we want to examine a Basic program byte by byte it is convenient to program one of the function keys to do this for us. Before continuing enter the following definition for function key 0 into your Electron:

```
*KEY 0 I%=PAGE:REPEAT:PRINT I%,?I%,TAB(
28);CHR$(?I%):I%=I%+1:UNTIL ?I%=255|M
```

This piece of code is essentially a loop which looks at every memory location starting at PAGE and displays the address of that location, the number stored at that location, and the corresponding character. This continues until a memory location is found that contains 255.

To see this in operation, create a very short Basic program. For example, type in the following two lines:

```
10 PRINT "HELLO"
20 END
```

If you now press Func/f0 you will see this program listed down the screen byte by byte. The first byte contains 13 as the starting point. The next two bytes, containing 0 and 10, represent the line number. You always multiply the value in the first byte by 256 and add on the value in the second byte to get the line number. The next byte always contains a count of the number of characters in this line of the program, in this case 14. Because the maximum value that can be stored in one byte of memory is 255 this sets the limit for the maximum length of any instruction in Basic.

After the line number and byte count you will find the bytes comprising this particular instruction which continues up to the next byte code of 13 marking the end of this instruction. In this case the first code is 32, the ASCII code for space. This is then followed by 241 which is the code or token for the PRINT instruction. Every Basic keyword is coded as a single byte in this way. We listed all the tokens in ELBUG Vol.1 No.7, and they are also contained in the Electron User Guide with the description of Basic. After the token for PRINT, you should see another 32 (another space) followed by the ASCII codes for the rest of the PRINT instruction. You should also be able to follow through the bytes of the second instruction with its token of 224 for END.

One peculiarity that you may notice is the blank line on the screen after the line containing the second byte of the line number (the value 10). This arises because the function key definition tries crudely to display every character on the screen. It tries to do this with the value 10 which is the ASCII code for Linefeed, and that's exactly what happens. This may also explain any other unexpected characters that appear on the screen. The function key definition given above is adequate rather than comprehensive.

Once you know how a Basic program is stored in the computer's memory it is not then too difficult to write a routine that goes through the program looking for spaces, REM statements and the like and removing these if necessary, and that's what the PACK program listed here does.

HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS

GETTING THE RIGHT CHARACTER - J.S. Swiszczowski

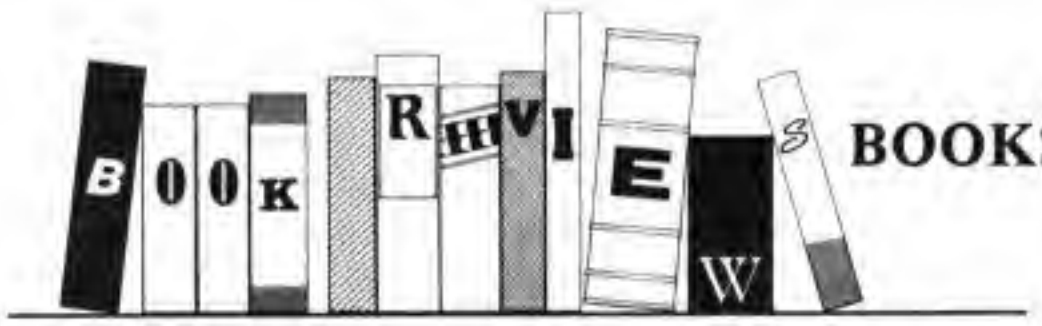
The following expressions will ensure that the correct character is interpreted on input ignoring the state of Caps Lock. This can make programs more user-friendly when the user may have selected Caps Lock mode, but the state of the micro is not known for certain. Each expression will input one character as shown.

Expression	Ensures correct entry of
key$=CHR$(GET OR &30)	digits
key$=CHR$(GET OR &60)	lower case
key$=CHR$(GET AND &5F)	upper case

LINE LISTING AFTER ERROR - P. Jollyman

If you use the following key definition, you can get a listing of the line in which the last error occurred. Once typed in this you can display any line which generates an error by pressing the keys Func and 1 together (Func-1).

```
*KEY1 C$="L."+STR$ERL+CHR$13:A%=138:X%=0:FORJ%=1TOLENC$:Y%=ASCMID$(A$,J%):
CA.&FFF4:N.|M          (Note that the '|' character is next to Break on the keyboard.)
```



BOOKS FOR PROGRAMMERS

Reviewed by Mike Williams

When you buy your Electron, you not only get a clear and readable User Guide, but a most valuable introduction to Basic programming in the form of the book "Start Programming with the Electron" by Masoud Yazdani. This is an excellent book to get you started, but where do you turn for more information?

Both of the books reviewed this month assume some knowledge of both machine and language so where better to look for that extra advice and help?

'Getting more from you BBC and Electron Computers' by Noel Kantaris & Keith Thompson, published by Sigma Technical Press at £6.95.

This is a most attractive looking book that has clearly attempted to take advantage of the high degree of compatibility between the Electron and the BBC micro. To a large extent I think that the book has been successful in this aim, as Teletext mode, one of the main features of the Beeb has been completely left out of this book. This is not to say that as an Electron owner you won't occasionally find references applicable only to that other machine, but that these are largely few in number and readily skipped. There are also some references to devices like printers which are only possible with the Electron if you buy one of the new printer interfaces (like the Plus 1 from Acorn reviewed in ELBUG issue 8).

Overall, this book provides a most thorough and comprehensive coverage of BBC Basic. That it does so in just over 200 pages means that the pace is quite fast, hence the desirability of having already started to program in a small way. The first three chapters introduce all the fundemantal aspects of Basic programming with a good introduction to high resolution graphics (using MOVE and DRAW) but a very sketchy description of SOUND and no reference at all to the use of envelopes. This is all good stuff, though I would wish more of the examples had a less mathematical flavour to them.

The next two chapters deal with strings, functions, procedures and subroutines. Again this is very thorough although, in my view, there is insufficient emphasis on the use of procedures and their role in producing well structured and readable programs.

The next two chapters deal with 'Advanced Graphics' covering much that is both useful and fundemental to graphics programming on the Electron. The emphasis is strongly on plotting high resolution graphics, and user defined characters, so vital in most games programs, get limited cover here. For example, there is nothing about how to build up larger characters (other than two side by side - the easiest arrangement to program), or about how to program two-coloured characters. On the other hand, the descriptions of the use of MOVE, DRAW and PLOT are quite comprehensive, and only a specialist book on graphics could be expected to contain more.

This book is well produced, the program listings are very clear, and there are 32 programs listed at the back as answers to exercises set throughout the earlier chapters. The book has a rather mathematical and academic flavour to it, and indeed it might well have been written as a textbook on Basic programming. If you have a technical bent and want a very thorough treatise on nearly all aspects

of BBC Basic then this could well be the book for you.

'Advancing with the Electron' by Peter Seal, published by Micro Press at £5.95

This is another most attractive looking book, though this time with only just over 100 pages. This is in some ways an unusual book in that it quite deliberately does not set out to teach Basic, but instead takes as its task the design, development and programming of a single major program for managing a small database. The design of large programs is something that beginners, and even some who are more than beginners, find a difficult task, and this is a most commendable attempt.

The first three chapters cover some basic programming concepts for the Electron, and then move on to discuss the design and development of the database system itself. This covers some of the necessary detail of file design and discusses the various functions to be included in the database program.

The main chapter in the book, indeed it takes up nearly half the book, is concerned with the progressive and detailed coding of the entire database program. For a number of reasons I feel less than enthusiastic about the result. I believe this section of the book would have been more readable had it been divided up functionally into three or four shorter chapters. The approach to coding seems to lack any real feel for structuring, and I often found myself lost as to exactly which part of the program I was dealing with, and more particularly how it related to the rest of the program.

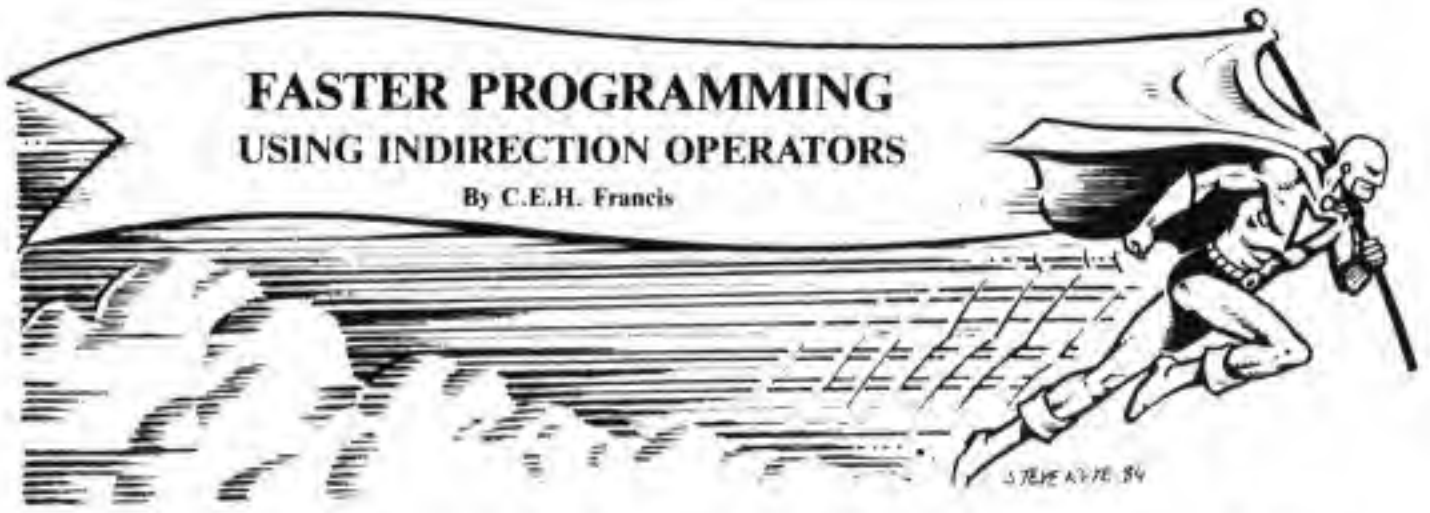
Developing a complete program in this way is a marvellous opportunity to really get to grips with the problems and techniques of program design and structure, and sadly this chance seems to have been missed.

I also found the detailed choice of particular coding techniques to be second best in several instances. I would have expected to see functions and procedures used almost exclusively throughout a program of this nature, but in fact subroutines are used as well. The author says, on page 35, "The main purpose of using procedures rather than subroutines is that if its necessary to shift them about the program, you can do so without having to reset line numbers. In this way it (viz, the procedure) is very similar to a data statement." I thought the merit in using procedures was that they were called by name with parameter passing, with the opportunity to produce a better structured and more readable program.

Frequent use of GOTO in many of the routines followed by a line number also hinders readability, and hence understanding, of the coding. Some detailed constructions are also confusing in printed form. On page 41 a PRINT statement contains (I think) 46 spaces. Surely it is much clearer, and takes up less space, to write SPC46 instead.

The last chapters in the book deal with testing, debugging and documentation. The information here is quite useful, but of a very general nature. It would have been interesting to discuss the design of specific test data to ensure that the program described performed according to the initial design and specification. Instead the book tends to concentrate more on how to debug programs typed in from book and magazine listings.

Overall I was rather disappointed with this book. The decision to base the entire book about the design and construction of one major program is a brave one on which the author is to be congratulated. It is a pity that the final result does not live up to its earlier promise. ●



In the July issue of ELBUG we published a game called 'Flowers of Hell'. In this article, one of a number of occasional articles on efficient programming, the author of this game divulges some of his secrets on making programs run faster.

If you have tried writing any but the most simple Basic game, the chances are that you will have tried using arrays (see also last month's article on 'Debugging Programs' for some more information about arrays). This causes two problems: arrays use up a lot of memory, and they are relatively slow. Fortunately both problems can be solved without resorting to machine code by the use of the so-called indirection operators provided by BBC Basic. The full description of indirection operators lies in the part of the manual that most of us don't read (pages 129, 130 and the section on assembly language programming), but in fact they are remarkably easy to use.

An indirection operator allows a Basic program to use directly any memory location. On many micros the actions involved are known as 'peeks' and 'pokes'. The simplest way to use indirection operators applies to integer arrays when it is known that all the numbers to be stored are positive integers less than 256. For example, two arrays could be used to store the X and Y co-ordinates of a user defined characters within a game, with one for the X values, and the other for the Y values. In this case the two arrays can be replaced directly. Consider the following short program, which loops round setting an array to a specific value:

```
10 DIM A%(999)
20 TIME=0
30 FOR I%=0 TO 999:A%(I%)=1:NEXT
40 PRINT TIME
```

In mode 6 this took 1.38 seconds on my Electron. Now replace A%(999) by A% 999 and A%(I%) by A%?I% (? is pronounced as 'query'). With this replacement the loop took 1.02 seconds: a saving of 25%. This can be a very major saving in a game in which you may access a number of arrays hundreds of times whenever the main loop of the program is executed. For example, in "Flowers of Hell" (July issue) by the time there are five fireballs the main loop has to read over two hundred array elements. The slowing down of "Flowers of Hell" is mainly due to the use of arrays (in this case slowing down doesn't matter as the game would otherwise become unplayable as the number of flowers and fireballs increases).

DIM A% 999 assigns 1000 bytes of memory from A% to A%+999. To find out where in memory this might be, type PRINT A% after the above program has been run. A%?I% gives the contents of memory location A%+I%, but this could also be accessed with ?(A%+I%). This gives the clue to an even greater saving. Replace line 30 above with:

```
30 FOR I%=A% TO A%+999:?I%=1:NEXT
```

This accomplishes exactly the same thing, but the program now runs in 0.92 seconds, a total saving of around 33%.

This is all very well, but what if you want to use an array which takes values requiring more than one byte of memory (i.e. integers greater than 255). The savings in time are no less dramatic using the ! (pronounced 'pling') indirection operator. The Electron stores integers in four consecutive bytes, and !I% gives the integer stored in locations I%, I%+1, I%+2, and I%+3. For an integer array with 1000 members, 4000 (1000*4) memory locations must be reserved, and the

step size should be 4. Replace lines 10 and 30 with:

```
10 DIM A% 3999
30 FOR I%=A% TO A%+3999 STEP4:!I%=1:
NEXT
```

This is barely slower, taking 0.94 seconds. In addition, any integer value can be placed in the location I% by using !I%=<value>.

String arrays can be replaced using the $ (pronounced 'string') indirection operator, and the savings are even greater. Try using

```
10 DIM A$(999)
30 FORI%=0 TO 999:A$(I%)="TEST":NEXT
```

This took 1.48 seconds on my Electron. When using indirection operators the string "TEST" requires five memory locations, one for each letter, and one for a Return to indicate the end of the string. To see how the $ indirection operator works try altering lines 10 and 30 to:

```
10 DIM A% 4999
30 FOR I%=A% TO A%+4999 STEP5:$I%="T
EST":NEXT
```

This saves over half a second, taking 0.96 seconds. Virtually no time is lost by setting aside more memory, and using a larger step size, for example DIM A% 9999 and STEP 10. The maximum length of a string which can be stored like this is one less than the step size used. Due to the way in which string arrays are allocated memory on the Electron, the savings in memory are quite significant.

Incidentally, when indirection operators are used like this some of the Basic string operators can be simulated very easily. In the above examples PRINT ?A% gives the ASCII code for T, while PRINT $A% gives 'TEST', as might be expected. PRINT $(A%+1) gives EST, simulating the operation of the RIGHT$ function. The Electron simply reads the string from wherever it is told to start, and stops when it reaches a Return (ASCII value 13). LEFT$ can also be simulated by placing a Return in a particular memory location. For example $(A%+2)="" which puts a Return character at A%+2, followed by PRINT $A% will give TE. Strings can be combined by placing the first letter of one string in the location occupied by the Return for another string. For example type $(A%+2)="STING". PRINT $A% now gives TESTING.

Finally it is worth examining the savings which can be made on using two (or more) dimensional arrays. These are even slower than one dimensional arrays. For example type in the lines listed here:

```
10 DIM A%(9,99)
30 FOR K%=0 TO 9:FOR J%=0 TO 99:
A%(K%,J%)=1:NEXT
```

When run, the program now takes 2.27 seconds. Indirection operators do not allow the use of more than one dimension, so the two dimensional array above must be replaced by the one dimensional set of memory locations DIM A% 999. Then K% and J% have to be replaced by I% running from A% to A%+999. This brings the time down to 0.92 seconds, but you do have to be careful with the arithmetic. In this example the actual location of the element A(K%,J%) is found as follows:

```
I%=A%+100*K%+J%
```

So you must remember that increasing I% by one corresponds to increasing J% by one, unless the value of J% is 99, when it corresponds to increasing K% by 1 and resetting J% to 0. Increasing I% by 100 corresponds to increasing K% by 1 and leaving J% alone. The situation is essentially similar in the case of the ! and $ indirection operators, but then you also have to be careful about step sizes. By the way, don't succumb to the temptation of letting the micro do the arithmetic for you by using an equation such as the one above inside the FOR-NEXT loop - you will very likely lose all the time saved!

THE MEMORY GAME

by Alan Dickinson

The Memory Game is based on the old card game of Pelmanism where you turn over cards one at a time, and match up pairs. This version makes excellent use of the Electron's colour graphics to provide an attractive variation of this game with a computing theme.

In our version of the Memory Game you are presented with eight rows of eight characters, all hidden from view. You can then look at any character to identify it, and by remembering positions, locate and match pairs of characters on the board.

The program listed below is in two parts, each of which must be typed in and saved on cassette. The first part consists entirely of character definitions which set up the various 'characters' used in the game. Type the lines in carefully, as any mistakes will show up in the design of the characters later on. The second program is loaded and run automatically by the first. You must make sure that the programs are recorded in the order presented here, and named as 'PEL1' and 'PEL2'.

You can choose between playing with four, eight or the full sixteen characters. Obviously, the more designs you play with, the harder the game. You will also need to select whether you want to play on your own, against another player or against the computer. The computer can play at three levels of skill, and on level three you'll find it hard to beat!

When you have selected the options you want, press the Return key or the space bar, and the game will begin. Enter your pairs by moving the hairline cursor using the cursor keys, and press Return when you're over the desired square. The program will briefly display each character selected. The computer will keep a record of your score, as well as how many tries you've had if you are playing alone. The game is over when all the squares are uncovered, and after the final scores are given you can play again by pressing any key.

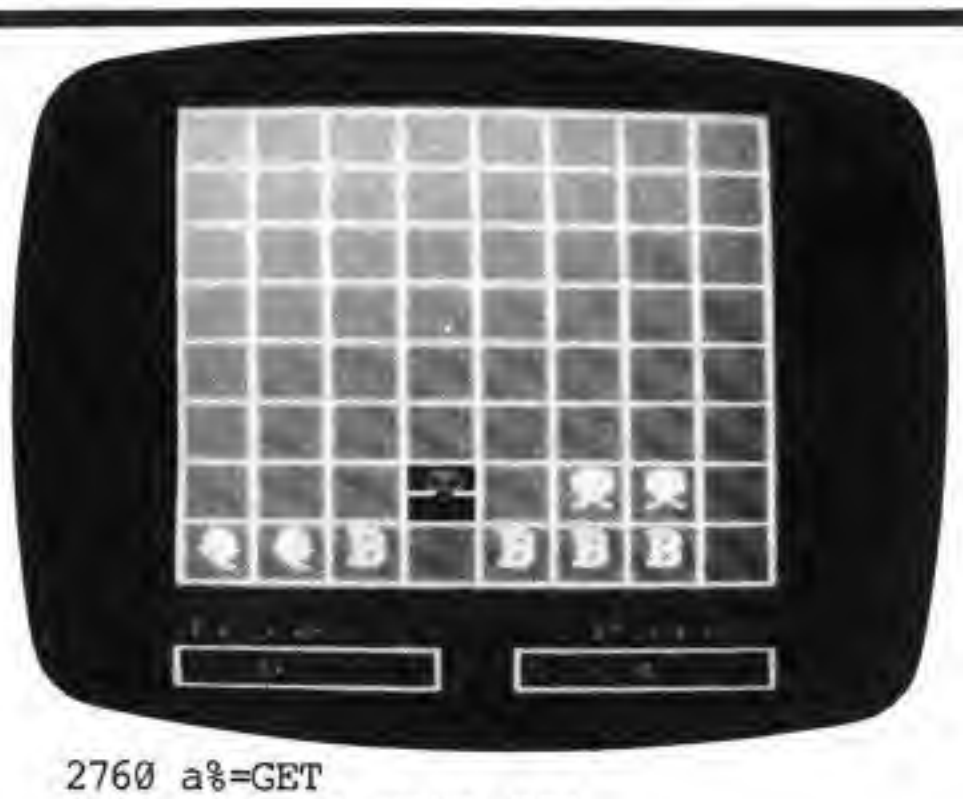
The very attractive use of user-defined characters makes this a most satisfying game to play, and illustrates how good presentation can add much to the quality of any computer game.

```
 10 REM PROGRAM Pelman 1
 20 REM VERSION E0.1
 30 REM AUTHOR A.Dickinson
 40 REM ELBUG OCT 1984
 50 REM PROGRAM SUBJECT TO COPYRIGHT
 60 :
100 REM Character Definitions
 110 REM 224-225 Acorn
 120 REM 226-227 B
 130 REM 228-229 Pacman
 140 REM 230-231 Vader
 150 VDU23,224,24,60,60,126,126,126,12
6,0,23,225,255,126,126,60,24,24,12,6
 160 VDU23,226,252,252,102,102,102,102
,124,124,23,227,102,102,102,102,252,252
,0,0
 170 VDU23,228,60,126,126,255,153,153,
153,153,23,229,255,255,171,213,255,255,
85,85
 180 VDU23,230,60,126,255,189,153,153,
255,255,23,231,126,60,24,24,36,66,129,0
 190 :
 200 REM 232-233 Diamond
 210 REM 234-235 Club
 220 REM 236-237 Heart
 230 REM 238-239 Spade
```

```
  240 VDU23,232,0,24,24,60,60,126,126,2
55,23,233,255,255,126,126,60,60,24,24
  250 VDU23,234,0,24,24,60,60,24,90,219
,23,235,255,255,90,24,24,24,60,126
  260 VDU23,236,0,102,231,255,255,255,2
55,255,23,237,126,126,126,60,60,24,0,0
  270 VDU23,238,0,24,60,60,126,126,126,
255,23,239,255,255,219,219,24,24,60,126
  280 :
  290 REM 240-241 Smiley
  300 REM 242-243 Grumpy
  310 REM 244-245 Train
  320 REM 246-247 Car
  330 VDU23,240,60,102,195,129,129,165,
165,129,23,241,129,165,165,189,129,195,
102,60
  340 VDU23,242,60,102,195,129,129,165,
165,129,23,243,129,189,165,165,129,195,
102,60
  350 VDU23,244,7,5,229,69,69,85,85,253
,23,245,253,255,255,254,255,3,171,168
  360 VDU23,246,0,0,62,34,34,34,34,227,
23,247,191,255,255,24,219,219,195,195
  370 :
  380 REM 248-249 Rook
  390 REM 250-251 Knight
  400 REM 252-253 Bishop
  410 REM 254-255 King
  420 VDU23,248,219,219,219,255,255,126
,126,126,23,249,126,126,126,126,255,255
,126,255
  430 VDU23,250,12,4,28,52,118,246,246,
254,34,23,251,30,28,60,126,255,255,126,
255
  440 VDU23,252,60,24,60,126,62,159,207
,231,23,253,247,255,126,60,255,255,126,
255
  450 VDU23,254,24,60,189,153,219,255,1
89,153,23,255,219,126,60,24,255,255,126
,255
  460 CLS:CHAIN"PEL2"
```

```
   10 REM PROGRAM Pelman 2
   20 REM VERSION E0.3
   30 REM AUTHOR  A Dickinson
   40 REM ELBUG OCT 1984
   50 REM PROGRAM SUBJECT TO COPYRIGHT
   60 :
  100 DIM M%(63)
  110 DIM N%(63)
  120 DIM O%(63)
  130 B%=0
  140 P$="1"
  150 H$="A"
  160 :
  170 ON ERROR IF ERR<>17 MODE6:PROCabe
nd:END
  180 REPEAT
  190 MODE6:PROCintro
  200 MODE5:PROCpelman
  210 MODE6:PROCsummary
  220 UNTIL FALSE
  230 :
 1000 DEF PROCheader
 1010 PRINTTAB(0,0)STRING$(40,"*")
 1020 PRINTTAB(0,4)STRING$(40,"*")
 1030 PRINTTAB(12,2);"The Memory Game";
 1040 ENDPROC
 1050 :
 1060 DEF PROCintro
 1070 VDU23,1,0;0;0;0;
 1080 PROCheader
 1090 PRINTTAB(13,6)"1. Solo"
 1100 PRINTTAB(13,7)"2. Two player"
 1110 PRINTTAB(13,8)"3. Computer(1)"
 1120 PRINTTAB(13,9)"4. Computer(2)"
 1130 PRINTTAB(13,10)"5. Computer(3)"
 1140 PRINTTAB(13,12)"A. 4 designs"
 1150 PRINTTAB(13,13)"B. 8 designs"
 1160 PRINTTAB(13,14)"C. 16 designs"
 1170 PRINTTAB(13,20)"Q. Quit"
 1180 PRINTTAB(13,24)"SPACE to start";
 1190 a$=P$:PROCoption
 1200 a$=H$:PROCoption
 1210 REPEAT
 1220 a$=GET$
 1230 PROCoption
 1240 UNTIL a$=" " OR ASC(a$)=13
 1250 ENDPROC
 1260 :
 1270 DEF PROCoption
 1280 IFa$="Q" CLS:PROCend:END
 1290 IFa$="1" P$="1":PROCv(6,6)
 1300 IFa$="2" P$="2":PROCv(7,6)
 1310 IFa$="3" P$="3":PROCv(8,6)
 1320 IFa$="4" P$="4":PROCv(9,6)
 1330 IFa$="5" P$="5":PROCv(10,6)
 1340 IFa$="A" H$="A":PROCv(12,12)
 1350 IFa$="B" H$="B":PROCv(13,12)
 1360 IFa$="C" H$="C":PROCv(14,12)
 1370 ENDPROC
 1380 :
 1390 DEF PROCv(j%,k%)
 1400 FORm%=k% TO k%+4
 1410 VDU31,6,m%,32,32,32
 1420 NEXT
 1430 VDU31,6,j%,45,45,62
 1440 ENDPROC
 1450 :
 1460 DEF PROCpelman
 1470 VDU19,0,14,0,0,0
 1480 VDU19,1,4,0,0,0
 1490 VDU19,2,6,0,0,0
 1500 VDU23,0,10,114;0;0;0;
 1510 PROCscreen
 1520 PROCsetboard
 1530 IFP$="1" PROCgame1 ELSE PROCgame2
 1540 PROCtune
 1550 TIME=0:REPEAT UNTIL TIME>350
```

```
1560 ENDPROC
1570 :
1580 DEF PROCtune
1590 RESTORE
1600 FOR i%=1 TO 7
1610 READ f%
1620 SOUND1,-12,f%+32,2:SOUND1,0,1,1
1630 NEXT
1640 SOUND1,-10,64,60
1650 DATA 64,72,48,56,64,72,88
1660 ENDPROC
1670 :
1680 DEF PROCscreen
1690 GCOL0,130:CLG
1700 GCOL0,1
1710 MOVE120,215:PLOT0,0,784
1720 PLOT81,1040,-784:PLOT81,0,784
1730 GCOL0,3
1740 FOR j%=1 TO 9
1750 MOVE128*j%,223:PLOT1,0,768
1760 MOVE128,127+j%*96:PLOT1,1024,0
1770 NEXT
1780 COLOUR1:COLOUR130
1790 FORx%=120 TO 716 STEP588
1800 MOVEx%,48
1810 PLOT1,0,64:PLOT1,448,0
1820 PLOT1,0,-64:PLOT1,-448,0
1830 NEXT
1840 ENDPROC
1850 :
1860 DEF PROCsetboard
1870 FORj%=0 TO 63
1880 N%(j%)=0:O%(j%)=0
1890 NEXT
1900 PROCtune
1910 IF H$="A" n%=4:m%=16
1920 IF H$="B" n%=8:m%=8
1930 IF H$="C" n%=16:m%=4
1940 FOR j%=0 TO n%-1
1950 FOR k%=1 TO m%
1960 Z%=RND(40)+RND(40)
1970 REPEAT
1980 Z%=Z%+1:IF Z%>63 Z%=0
1990 UNTIL N%(Z%)=0
2000 N%(Z%)=1:M%(Z%)=224+j%*2
2010 NEXT
2020 NEXT
2030 ENDPROC
2040 :
2050 DEF PROCgame1
2060 C%=0:X%=0:Y%=0:G%=0
2070 COLOUR1:COLOUR130
2080 PRINTTAB(2,27)"Tries"
2090 PRINTTAB(13,27)"Pairs"
2100 REPEAT
2110 PRINTTAB(4,29);G%;
2120 PRINTTAB(14,29);C%;
2130 T%=FNpair
2140 G%=G%+1
2150 UNTILC%=32
2160 ENDPROC
2170 :
2180 DEF PROCgame2
2190 C%=0:X%=0:Y%=0:L%=0:R%=0:G%=TRUE
2200 COLOUR1:COLOUR130
2210 REPEAT
2220 PRINTTAB(8,29);
2230 PRINTTAB(2,27)SPC(7);
2240 PRINTTAB(11,27)SPC(7);
2250 IFG% PRINTTAB(2,27)"Player1";
2260 IFNOTG%PRINTTAB(11,27)"Player2";
2270 PRINTTAB(3,29);L%;
2280 PRINTTAB(13,29);R%;
2290 T%=FNpair
2300 IF G% L%=L%+T% ELSER%=R%+T%
2310 IF T%=0 G%=NOT G%
2320 UNTIL L%+R%=32
2330 ENDPROC
2340 :
2350 DEF FNpair
2360 REPEAT
2370 F%=FNchoice(1)
2380 UNTIL N%(F%)=1
2390 PROCsquare(F%,0,1)
2400 REPEAT
2410 S%=FNchoice(2)
2420 UNTIL N%(S%)=1 AND S%<>F%
2430 PROCsquare(S%,0,1)
2440 IFM%(F%)=M%(S%) T%=1 ELSE T%=0
2450 IF T%=1 PROCmatch
2460 TIME=0:REPEAT UNTIL TIME>100
2470 PROCsquare(F%,1,T%)
2480 PROCsquare(S%,1,T%)
2490 =T%
2500 :
2510 DEF PROCmatch
2520 FORj%=1 TO 200 STEP32
2530 SOUND1,-9,j%,1
2540 NEXT
2550 N%(F%)=0:N%(S%)=0:C%=C%+1
2560 ENDPROC
2570 :
2580 DEF FNchoice(ch%)
2590 IFP$>"2" IFNOTG% THEN=FNcomp(ch%)
2600 GCOL4,0
2610 REPEAT
2620 MOVEX%*128+128,Y%*96+268
2630 PLOT1,128,0
2640 a$=FNin:SOUND&11,-10,100,1
2650 PLOT1,-128,0
2660 IFa$="L" IFX%>0 X%=X%-1
2670 IFa$="R" IFX%<7 X%=X%+1
2680 IFa$="D" IFY%>0 Y%=Y%-1
2690 IFa$="U" IFY%<7 Y%=Y%+1
2700 UNTIL a$="E"
2710 =X%+Y%*8
2720 :
2730 DEF FNin
2740 *FX4,1
2750 *FX15
```



```
2760 a%=GET
2770 IFa%=13  THEN="E"
2780 IFa%=136 THEN="L"
2790 IFa%=137 THEN="R"
2800 IFa%=138 THEN="D"
2810 IFa%=139 THEN="U"
2820 GOTO2760
2830 :
2840 DEF FNcomp(ch%)
2850 TIME=0:REPEAT UNTIL TIME>150
2860 IFP$="3" ORch%=1 r%=FNcomp3
2870 IFP$>"3" IFch%=2 r%=FNcomp4
2880 SOUND&11,-8,r%*3,3
2890 =r%
2900 :
2910 DEF FNcomp3
2920 r%=RND(64)
2930 REPEAT
2940 r%=r%+1:IFr%>63r%=0
2950 UNTIL N%(r%)=1 AND r%<>F%
2960 =r%
2970 :
2980 DEF FNcomp4
2990 IFP$="4" IFRND(10)<7 THEN=FNcomp3
3000 k%=99
3010 FOR j%=1 TO 63
3020 IF N%(j%)=1 AND O%(j%)=1 AND j%<>
F% AND M%(j%)=M%(F%) THEN k%=j%:j%=99
3030 NEXT
3040 IFk%<99 THEN=k% ELSE=FNcomp3
3050 :
3060 DEF PROCsquare(z%,c%,d%)
3070 REM z - square number
3080 REM c - background colour
3090 REM d - detail (0=no,1=yes)
3100 O%(z%)=1
3110 GCOL0,c%
3120 x%=(z%MOD8)*128+136
3130 y%=(z%DIV8)*96+312
3140 MOVEx%,y%:PLOT0,116,0
3150 PLOT81,-116,-88:PLOT81,116,0
3160 IFd%=0 ENDPROC
3170 IFc%=0 GCOL0,1 ELSE GCOL0,3
3180 MOVE x%+32,y%-8
3190 VDU5,M%(z%),10,8,M%(z%)+1,4
3200 ENDPROC
3210 :
3220 DEF PROCsummary
3230 PROCheader
3240 IF P$="1" PROCsu1 ELSE PROCsu2
3250 *FX15
3260 A$=GET$
3270 ENDPROC
3280 :
3290 DEF PROCsu1
3300 P%=3200/G%+0.5:IFP%>B% B%=P%
3310 PRINTTAB(13,8)"Percentage"
3320 PRINTTAB(15,10);P%;" %"
3330 PRINTTAB(13,15)"High score"
3340 PRINTTAB(15,17);B%;" %"
3350 ENDPROC
3360 :
3370 DEF PROCsu2
3380 PRINTTAB(13,8)"Player 1 scored"
3390 PRINTTAB(15,10);L%
3400 PRINTTAB(13,15)"Player 2 scored"
3410 PRINTTAB(15,17);R%
3420 ENDPROC
3430 :
3440 DEF PROCabend
3450 ON ERROR OFF:CLS
3460 REPORT
3470 PRINT" at ";ERL
3480 PROCend
3490 ENDPROC
3500 :
3510 DEF PROCend
3520 *FX4
3530 *FX15
3540 ENDPROC
```

HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS HINTS

SCREEN COLOUR CHANGING - A. Porter

You can change the colours for screen displays without using the VDU 19 sequence by using the Control key: type in Control-S then press the key that corresponds to the logical colour, then the key for the physical colour, then three '0's. For example, to change the background colour (0) to blue, type Ctrl-S, followed by 04000.

Subscriptions and Software Help Line
St Albans (0727) 60263
Manned Mon-Fri 1pm-4pm.

BACK ISSUES AND SUBSCRIPTIONS

BACK ISSUES (Members only)

All back issues will be kept in print (from November 1983). Send 90p per issue PLUS an A5 SAE to the subscriptions address. Back copies of BEEBUG are available to ELBUG members at this same price. This offer is for members only, so it is ESSENTIAL to quote your membership number with your order. Please note that the advertising supplements are not supplied with back issues.

Subscription and Software Address

ELBUG
PO BOX 109
High Wycombe
Bucks

SUBSCRIPTIONS

Send all applications for membership, and subscription queries to the subscriptions address.

MEMBERSHIP COSTS:

U.K.
£5.90 for 6 months (5 issues)
£9.90 for 1 year (10 issues)
Eire and Europe
Membership £16 for one year.
Middle East £19
Americas and Africa £21
Elsewhere £23
Payments in Sterling preferred.

SOFTWARE (Members only)

This is available from the software address.

MAGAZINE CONTRIBUTIONS AND TECHNICAL QUERIES

Please send all contributions and technical queries to the editorial address opposite. All contributions published in the magazine will be paid for at the rate of £25 per page.

We will also pay £10 for the best Hint or Tip that we publish, and £5 to the next best. Please send all editorial material to the editorial address opposite. If you require a reply it is essential to quote your membership number and enclose an SAE.

Editorial Address

ELBUG
PO Box 50
St Albans
Herts

ELBUG MAGAZINE is produced by BEEBUG Publications Ltd.
Editor: Mike Williams.
Production Editor: Phyllida Vanstone.
Technical Assistants David Fell, John Waller and Alan Webster.
Managing Editor: Lee Calcraft.
Thanks are due to, Sheridan Williams, and Adrian Calcraft for assistance with this issue.

New Elbug Binders

We have produced an attractive hard-backed binder for the ELBUG magazine. These binders are green in colour with "ELBUG" in gold lettering on the spine and allow for the whole of one volume of the magazine to be stored as a single reference book.

Each binder will accommodate 10 ELBUG magazines, and is supplied with 12 wires to enable the index and the latest copy of the supplement to be included within the binder if required. Individual issues may be easily added and removed, allowing for the latest volume to be filed as it arrives.

The price of the new ELBUG binder is £3.90 including VAT, please add 50p post and packing for delivery within the U.K. Overseas members please send the same amount, this will cover the extra postage but not VAT.
Plese send to:
BEEBUGSOFT, PO BOX 109, High Wycombe, Bucks, HP10 8HQ.

THE BEST OF ELBUG ON CASSETTE

Many of the best programs published in ELBUG have been collected together and published by Penguin Books under the name "Games and other programs for the Acorn Electron" at £3.95. This book is part of the Penguin Acorn Computer Library and at present there is just one other title available though others are planned.

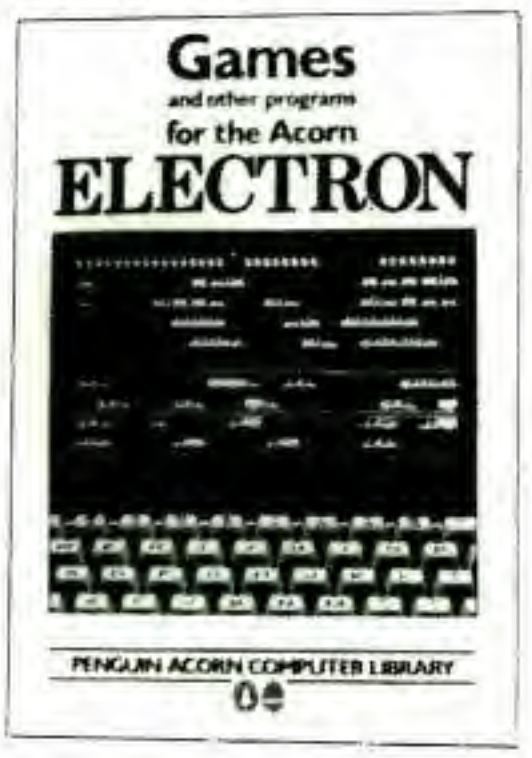
There are 20 programs in all in four different categories:

Action Games
- Munch-Man
- Mars Lander
- Invasion
- Robot Attack
- Hedgehog

Thought games
- Higher/Lower
- Five-Dice
- Life
- Anagrams
- Return of the Diamond

Visual Displays
- Union Jack
- Square Dance
- Ellipto
- Screenplay
- 3-D Rotation

Utilities
- Sound Wizard
- Bad Program Lister
- 3-D Lettering
- Bad Program Rescue
- Double Height Text



All 20 programs are now available on cassette from our software address (in High Wycombe) price £7 to members and £9 to non-members, plus 50p post & packing in either case.

ELBUG MAGAZINE CASSETTE

To save wear and tear on fingers and brain, we offer, each month, a cassette of the programs featured in the latest edition of ELBUG. The first program on each tape is a menu program, detailing the tape's contents, and allowing the selection of individual programs. The tapes are produced to a high technical standard by the process used for the BEEBUGSOFT range of titles.

Magazine cassettes have been produced for each issue of ELBUG from Volume 1 Number 1 onwards and are all available from stock, priced £3.00 each inclusive of VAT. See below for ordering information.

This months cassette includes:

Volume 1 Number 10
Digger (an action packed arcade-style game), Fireworks Display, a useful program compactor, The Memory Game (an updated version of Pellmanism), five example programs illustrating Electron Graphics, plus the winning program in the 'Oddfactors' Brainteaser competition, and as an extra attraction, another colourful action game, Astro Wars, written in machine code.

MAGAZINE CASSETTE SUBSCRIPTION

We are also able to offer ELBUG members subscription to the magazine cassette, this gives the added advantage of receiving the cassette at around the same time as the magazine each month. Subscriptions may either be for a period of 1 year or 6 months. (NOTE Magazine cassettes are produced 10 times each year).

If required, subscriptions may be backdated as far as Volume 1 Number 1, so when applying please write to the address below quoting your membership number and the issue from which you would like your subscription to start.

MAGAZINE CASSETTE ORDERING INFORMATION

Individual ELBUG Magazine Cassettes £3.00.
P & P: Please add 50p for the first and 30p for each subsequent cassette.
Overseas orders: Please send the same amount, this will include the extra post but not VAT.

Magazine Cassette Subscription

1 YEAR (10 issues)	£33.00 Inclusive	.....	O'SEAS	£39.00 No VAT payable
6 MONTHS (5 issues)	£17.00 Inclusive	.....	O'SEAS	£20.00 No VAT payable

Please be sure to specify that you require subscription to the ELBUG magazine cassette (as opposed to the BEEBUG cassette), and enclose your membership number with a cheque made payable to BEEBUGSOFT.

Please send to..

ELBUG Magazine Cassette, BEEBUGSOFT, PO Box 109, High Wycombe, HP10 8HQ

Printed in England by Staples Printers St Albans Limited at The Priory Press

ELBUG Magazine Cassette

Each month we offer a cassette containing the programs featured in the latest issue of ELBUG and overleaf is a list of the programs included on each tape from Volume 1 Number 1 up to this month's issue. All tapes are available from stock and are ready for immediate despatch, price £3.00 including VAT.

Cassette Subscription

We are also able to offer subscription to the magazine cassette for periods of 6 months (5 issues) or 1 year (10 issues), prices are shown below. Subscribers have the added advantage of receiving their cassettes generally just before, or at the same time as the magazine.

SPECIAL OFFER

If you take out a new subscription to the ELBUG magazine Cassette before November 16th using the form below, you may claim any one of the past magazine cassettes absolutely free.

MAGAZINE CASSETTE ORDER FORM

			Number	Price	Total
Volume 1	Number	1	.....	£3.00	.....
Volume 1	Number	2	.....	£3.00	.....
Volume 1	Number	3	.....	£3.00	.....
Volume 1	Number	4	.....	£3.00	.....
Volume 1	Number	5	.....	£3.00	.....
Volume 1	Number	6	.....	£3.00	.....
Volume 1	Number	7	.....	£3.00	.....
Volume 1	Number	8	.....	£3.00	.....
Volume 1	Number	9	.....	£3.00	.....
Volume 1	Number	10	.....	£3.00	.....
MAGAZINE CASSETTE SUBSCRIPTION UK 6 MONTHS				£17.00	.....
MAGAZINE CASSETTE SUBSCRIPTION OVERSEAS 6 MONTHS				£20.00	.....
MAGAZINE CASSETTE SUBSCRIPTION UK 1 YEAR				£33.00	.....
MAGAZINE CASSETTE SUBSCRIPTION OVERSEAS 1 YEAR				£39.00	.....
Postage...Please add 50p for first cassette and 30p for each subsequent item.				**POSTAGE**	.....
Subscription rates include post				**TOTAL**	=====

ELBUG MAGAZINE CASSETTE SUBSCRIPTION

SUBSCRIPTION TO RUN FROM .....

STATE WHICH FREE CASSETTE
(IF ORDERED BEFORE 16th NOVEMBER) .....

NAME .....

ADDRESS .....

.....

SEE OVER FOR MAGAZINE CASSETTE CONTENTS LIST

Send to: – ELBUG, PO Box 109, High Wycombe, Bucks. HP10 8HQ

ELBUG MAGAZINE CASETTES CONTENTS LIST

Volume 1 Number 1

Munch-Man – a Snapper type of game, Sound Wizard for designing and experimenting with sound envelopes, Graphics example program, a utility for producing double height characters, Highlo Card Game, a colourful Union Jack display, A Keyset program for setting up the function keys, and the exciting Hedgehog game.

Volume 1 Number 2

Return of the Diamond – a fascinating adventure game, a utility for displaying 3D lettering, an interesting visual display called Square Dance, ASTAAD – a versatile computer aided design program, a musical Christmas card, Robot Attack game, a Graphics example program, Santa's Parcels game, a utility to rescue 'Bad Programs', and a fast moving football game.

Volume 1 Number 3

Mars Lander game, a program for rotating, enlarging and reducing 3D objects, examples of Electron Graphics (3), a utility for designing a new character set, Reversi board game, a changing visual pattern based on ellipses, a utility for listing 'Bad Programs', and a challenging Dive Bomber game.

Volume 1 Number 4

Killer Dice game, The Spider and the Fly – an amusing visual display, further examples of Electron Graphics (7), Moving Chequer Board display, a versatile editor for developing sound envelopes, and the superb Block Blitz game.

Volume 1 Number 5

Invasion of the Aliens game, more examples of Electron Graphics (6), a short utility for saving screen displays, a simulation of continually changing fabric patterns, a classic Dominoes game, a versatile Utility Editor for Basic programmers, and the exhilarating Elevasion game.

Volume 1 Number 6

Hunt the Numers game, Invisible Alarm Clock, a Selective Renumber utility for basic programs, ASTAAD2 – the original CAD program extended, Graphics example programs (3), Lunar Escape game, Dancing Lines – an interesting visual display, and Four in a Row game.

Volume 1 Number 7

Flip Fap Game, Screen Freezer utility, routines for Expanding and Rotating Characters, Cursor keys demonstration, an interesting Stock Market Game, an action packed Galactic Invasion Game, and Niagara, the winning Electron entry in the March Brainteaser competition.

Volume 1 Number 8

The Flowers of Hell game, Disco Lights Display, automatic Cassette Indexer, the classic Breakout game for the Electron, Graphics Example (on manipulating colours), a useful Program Protection routine, and a colourful version of the Pontoon card game.

Volume 1 Number 9

Codebreaker game, Harmonograph display, a Mini Text Editor for simple word processing, Build a House graphics display, Kayak Kapers game, graphics demonstration program – GCOLMANIA, Dartboard game and a program to generate a TV/monitor Test Card.

Volume 1 Number 10

Digger (an action packed arcade-style game), Fireworks Display, a useful program compactor, The Memory Game (an updated version of Pellmanism), five example programs illustrating Electron Graphics, plus the winning program in the 'Oddfactors' Brainteaser competition, and as an extra attraction, another colourful action game, Astro Wars, written in machine code.